PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : **H04L 12/00** | **A2**

(11) International Publication Number: **WO 99/53648**

(43) International Publication Date: 21 October 1999 (21.10.99)

(21) International Application Number: PCT/US99/07808

(22) International Filing Date: 9 April 1999 (09.04.99)

(30) Priority Data:
09/058,597 10 April 1998 (10.04.98) US

(71) Applicant: TOP LAYER NETWORKS, INC. [US/US]; 2400 Computer Drive, Westboro, MA 01581–1770 (US).

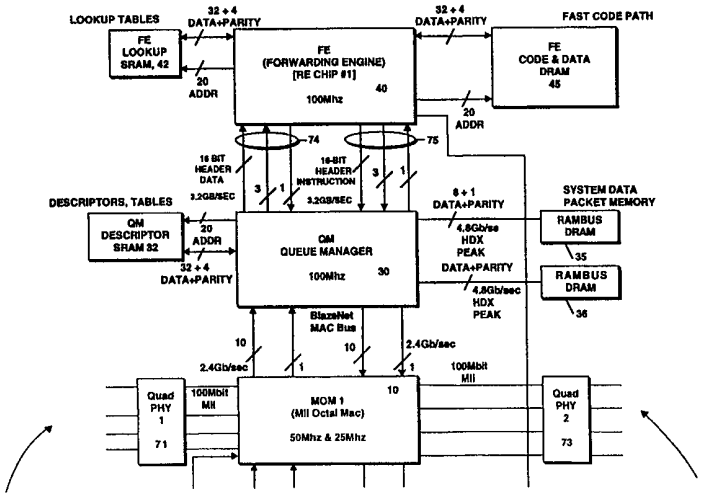
(72) Inventors: ROBINS, Cary, B.; Top Layer Networks, Inc., 2400 Computer Drive, Westboro, MA 01581–1770 (US). NARAYANASWAMY, Krishna; Top Layer Networks, Inc., 2400 Computer Drive, Westboro, MA 01581–1770 (US). ROSS, Theodore, L.; Top Layer Networks, Inc., 2400 Computer Drive, Westboro, MA 01581–1770 (US). SPINNEY, Barry, A.; Top Layer Networks, Inc., 2400 Computer Drive, Westboro, MA 01581–1770 (US).

(74) Agents: CHOW, Stephen, Y. et al.; Perkins, Smith & Cohen, LLP, 30th floor, One Beacon Street, Boston, MA 02108 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: SYSTEM AND PROCESS FOR HIGH-SPEED PATTERN MATCHING FOR APPLICATION-LEVEL SWITCHING OF DATA PACKETS



(57) Abstract

A process and system for switching connections of data packet flows between nodes of data processing system networks operating on diverse protocols according to the application layer information on the data packets. The process retrieves and hashes the header information to from an index into memory where a flow tag pointer is stored. The flow tag points to flow switching information that directs the forwarding of the packet. The switching information is sent along with the packet data to direct the forwarding state information about the flow is updated in the flow switching information. The hash function includes a multiplication and division by polynomials forming a hash result and a signature result. Both hash and signature are used to ensure that the information retrieved is valid. If invalid, the prehashed header information is parsed to determine the forwarding information. This forwarding information is stored for later use and the appropriate flow tag pointer is stored in the hash result index.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

SYSTEM AND PROCESS FOR HIGH-SPEED PATTERN MATCHING FOR APPLICATION-LEVEL SWITCHING OF DATA PACKETS

CROSS REFERENCE

This application is being filed in conjunction with applications for United States Patents for "System and Process for Application-level Connection of Data Processing Networks" by Barry Spinney et al., "High-Speed Data Bus for Network Switching" by Nigel Poole, and "System and Process for Flexible Queuing of Data Packets in Network Switching" by Barry Spinney et al. which have substantially similar disclosures and are assigned to a common entity, Top Layer Networks, Inc., formerly called BlazeNet, Inc.

BACKGROUND OF THE INVENTION

The field of the invention is that of the networking of data processing systems or computers and, more particularly, that of the switchable connection of Local Area Networks ("LANs") such as those supported by the Ethernet protocol and Wide Area Networks ("WANs") such as those supported by the Asynchronous Transfer Mode ("ATM") protocol.

One of the major problems in the field of connecting networks is that the variety of different network protocols used to communicate between different data processing systems on particular networks makes communication between such networks difficult. Another major problem is that most network protocols require considerable configuration of parameters when adding computer systems or nodes, typically accomplished by manual input of device addresses by network professionals who nonetheless make mistakes. This problem may be exacerbated when connecting across network boundaries.

Current connection of networks, including the mechanisms used to connect the Internet, is accomplished using devices known as "bridges" and "routers." Roughly speaking, bridges connect different networks at the "data link" layer or Layer 2 of the OSI Network model, *see* Schwartz, Mischa, *Telecommunication Networks* at 75-99 (Addison-Wesley 1987), and routers connect different networks at the "network" layer or Layer 3 of the OSI model. A packet of data is preceded by

headers corresponding to layers of communication, with the first in time header corresponding to the lowest Layer 1, the physical link, and proceeding up to Layer 7, the application layer (other models have fewer layers. The "application layer" here refers to functions at Layers 5-7 of the OSI model). When packets of information are received at a bridge, the bridge processor forwards the packet on a data link according to the information in the data link header (following the physical link header). When packets of information are received at a router, the packet is routed according to the information in the network header. These headers, however, do not contain information about the quality of service required by the application to which the data packet pertains; thus, each packet is forwarded according to the data link or network protocol which may or may not include a priority flag, typically for network management operations.

The types of applications requiring data transmission on current networks call for a wide range of service. Thus, in communications with a file server, requests uploaded from a client for downloading of data require relatively little bandwidth, while downloading of massive amounts of data requires great bandwidth to be accomplished in a reasonable time. Streaming of audio-visual ("multimedia") information requires guaranteed bandwidth at regular intervals to avoid perceivable interruptions or "jitter". E-mail, file server requests, HTTP, word processing each have their own application protocols with associated header information that can be associated with their communication needs, including bandwidth.

Network switching schemes that consider information above the network layer, so-called "Layer 4 switches," are just coming on the market and appear typically to involve software implementations that are slow and only consider a portion of the Layer 4 or transport layer header (the "TCP" part of TCP/IP or transport control protocol/internetwork protocol).

SUMMARY OF THE INVENTION

The present invention connects networks at the application layer and uses more information above Layer 3 of the OSI model than other network switches in the present state of the art.

The invention performs "flow switching" or connection, wherein, based on the information in a received data packet at Layer 4 and above, the invention identifies a "flow," that is, a sequence of network messages that occur as a result of a requested process such as reading a file, sending an e-mail message, browsing a web site, initiating a file transfer, making a database query, etc., and routes the packet accordingly, thereby establishing a "virtual connection" at Layer 4 and above. The invention is further adapted for "application flow switching," wherein the invention classifies received frames into flows based not only on the Layer 2 MAC or Layer 3 network address, but also on the information contained in higher layers, even up to "Application" Layer 7 of the OSI model. Thus, the invention can differentiate between flows that result from web browsing and flows that result from a file transfer or database query, even though both may use the same Layer 3 protocol.

In the preferred embodiment, this is accomplished using a combination of hardware and software optimized for speed or for flexibility at their respective functions. Thus, dedicated "silicon" or gates at the chip level are employed to extract rapidly information from the data link headers corresponding to the relatively few data link protocols such as Ethernet, Fast Ethernet, and Frame Relay, and from the network headers of the relatively few network protocols such as Internet (IPv4, IPX, IPv6), SNA, and DECNet, while application protocols in up to 128 bytes of header information are recognized by fast pattern matching software. By looking at the application header, the switch can make "intelligent" decisions about quality of service to be applied to a particular flow or stream of packets (such as e-mail, which is priority-based, as opposed to multimedia, which is bandwidth-guarantee-based) and can keep all connections while backing off of all applications fairly.

By using internally standard or "canonical" headers including data link and network information deduced or inferred at the port interfaces, and comparing hashed versions of the canonical headers to identify the packets to flows with common flow rules, the invention very efficiently establishes a virtual connection between the appropriate ports associated with a given

flow. This feature allows the system to be "frame or cell"-independent and to route ATM traffic as not heretofore done.

In the preferred embodiment, thousands of transmit queues are possible (pointing to data packets in fast storage) that allow thousands of connections as well as different qualities of service to be attached to individual queues.

The "intelligence" of the system in tracking packets according to the flow allows "cut through" flow, that is, the output from a port of portions of a data packet stream even as portions of the data packet stream are entering a port. Many other intelligent functions are possible because of the flexible and scalable architecture of the system using interface ASICs (application-specific integrated circuits) to "canonicalize" Layer 2 and 3 header information, a high speed bus, a queue manager ASIC which rapidly implements queuing decisions of a fast relay engine ASIC, and a background engine ASIC that monitors the flow connections.

DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a preferred embodiment of the invention.

FIG. 2 is a block diagram of the media interface ASIC (MOM) of the preferred embodiment of the invention.

FIG. 3 is a block diagram of the queue manager ASIC (QM) of the preferred embodiment of the invention.

FIG. 4 is a block diagram of the relay engine ASIC (RE) of the preferred embodiment of the invention.

FIG. 5 is a schematic diagram of the data flow of the preferred embodiment of the invention.

FIG. 6 is a general flow diagram of the processes used in the preferred embodiment of the invention.

FIG. 7A shows the data structure of a canonical header used in the preferred embodiment of the invention.

FIG. 7B shows the data structure of a portion of the canonical header used in the preferred embodiment of the invention.

FIG. 7C shows the data structure and possible entries of another portion of the canonical header used in the preferred embodiment of the invention.

FIG. 7D shows the data structure and possible entries of another portion of the canonical header used in the preferred embodiment of the invention.

FIG. 8A shows the data structure of another portion of the canonical header used in the preferred embodiment of the invention.

FIG. 8B shows the data structure and possible entries of another portion of the canonical header used in the preferred embodiment of the invention.

FIG. 8C shows the data structure and possible entries of another portion of the canonical header used in the preferred embodiment of the invention.

FIG. 9 is a block diagram of the high-speed bus used in the preferred embodiment of the invention.

FIG. 10 shows the differential character of the bus lines of FIG. 9.

FIG. 11 is a schematic of a transmit circuit used on the bus shown in FIG. 9.

FIG. 12 is a timing diagram of the transmit circuit of FIG. 11.

FIG. 12A is a composite timing of the transmit circuit of FIG. 11.

FIG. 13 is a schematic of a clock delay circuit used in transmission on the bus shown in FIG. 9.

FIG. 13A is a timing diagram of signals on the circuit shown in FIG. 13.

FIG. 14 is a detail of the circuit shown in FIG. 13.

FIG. 15 (Table 1) shows the possible values and meanings of a control bit used in the bus shown in FIG. 9.

FIG. 16 shows a sequence of control bits shown in FIG. 15.

FIG. 17 is a block diagram showing the token ring arbitration used between the interface chips shown in FIG. 1.

FIG. 18 shows a sequence of cell transmissions used in the preferred embodiment of the invention.

FIG. 19 shows a pointer-register structure used in the preferred embodiment of the invention.

FIG. 20A shows the data structure of a hash table entry used in the preferred embodiment of the invention.

FIG. 20B shows the data structure of another hash table entry used in the preferred embodiment of the invention.

FIG. 21 is a timing diagram for the control signals used on the bus shown in FIG. 9.

FIG. 22 shows possible values and meanings of control bits used on the bus shown in FIG. 9.

FIG. 23 shows as an example a sequence of control bits that may be seen on the bus shown in FIG. 9.

FIG. 24 shows diagrammatically the cell transmissions for possible cells transmitted on the bus shown in FIG. 9.

FIG. 25 shows the possible values and meanings for codes used on the bus shown in FIG. 9.

FIG. 26 shows the data structure of a field of the canonical header used in the preferred embodiment at different times.

FIG. 27 shows details of the data structure of one of the subfields shown in FIG. 26.

FIG. 28 shows the data structure of a temporary "burst" header used in the preferred embodiment of the invention.

FIG. 29 shows a set of linked descriptors mapped to a data packet used in the preferred embodiment.

FIG. 29B shows a set of linked descriptors used in the preferred embodiment to describe an incomplete packet.

FIG. 30 shows the linking of descriptors used in the preferred embodiment to establish virtual queues.

FIG. 30B shows the linking to buffer descriptors of receive and transmit context tables used in the preferred embodiment to track data cells forming a packet.

FIG. 31 is a representation of a credit-managed transmission system used in the preferred embodiment of the invention.

FIG. 32 is a representation of a ring-pointer system used in the preferred embodiment of the invention to determine whether credits should be issued in the system represented in FIG. 31.

FIG. 33 is a more detailed representation of the system represented in FIG. 31.

FIG. 34 is a representation of a hierarchical queue system used in a preferred embodiment of the invention.

FIG. 35 shows the data structure of a transmit context table entry used in a preferred embodiment of the invention.

FIG. 35A shows the data structure of a field of the data structure shown in FIG. 35.

FIG. 35B shows the possible service policies encoded in the Q SVC Policy field of the data structure shown in FIG. 35.

FIG. 36 shows the data structure of the queue table used in the preferred embodiment.

FIG. 37 represents possible links and queues in the transmission phase of the preferred embodiment.

FIG. 38 shows the operation of the standby scheduler used in a preferred embodiment of the invention.

FIG. 39A represents a linked descriptor set representing a complete packet in memory in the preferred embodiment.

FIG. 39B represents the delinking of the descriptor set shown in FIG. 39A to free the buffers described by the linked descriptor set shown in FIG. 39A.

FIG. 40 is a block diagram of a DRAM control system used in the preferred embodiment of the invention.

DETAILED DESCRIPTION

The specification will be organized as follows:

1. BlazePath™/BlazeFire™ Architecture/Chip Set
2. Header "Canonicalization" and Packet "Cellularization"
3. BlazeWire™ High-Speed MAC Bus
4. Data Flow In
5. Queue Pointer Management and Operation
6. Relay Engine Operations/Flow Matching (FastPath™)
7. Transmission Scheduling
8. Download to Interfaces/Transmission Credit Loops
9. Ultra-High Speed RAMBUS® Operation
10. Background Engine/Initialization/Monitoring

1. BlazePath™/BlazeFire™ Architecture/Chip Set

The architecture of the invention, called the BlazePath™ architecture, comprises application layer flow switching or

connection performed by virtually allocating (by pointers to high speed data buffers) incoming data packets to one or more of a large number of virtual queues according to decisions made on the basis of information in the application header of the packets characterizing their membership in a particular flow. To enhance the throughput or bandwidth of the system, a preferred embodiment, the AppSwitch™ application flow switch, makes decisions according to the initial packet(s) of the flow and matches a hashed version of the header information to identify subsequent packets of the flow. By "canonicalizing" the header information of the incoming flow and splitting lengthy frames into smaller internal cells (but keeping them logically connected), the system is "cell or frame" independent.

Referring to Fig. 1, in a preferred embodiment, the architecture is implemented in the BlazeFire™ chipset connected by the BlazeWire™ MAC Bus 60. The architecture is centered around a 287k-gate Queue Manager ("QM") ASIC 30 operating at 100 MHz which implements the queue-pointer scheme allowing 16,000,000 queues for pointers (24-bit definition) to locations in a high-speed data buffer 35 and 36 connected to the QM 30 in which incoming packets are temporarily stored. The queues are loaded based on decisions made by the 410k-gate Relay Engine ("RE") or Forwarding Engine ("FE") ASIC 40 operating at 100 MHz which includes an Argonaut RISC (ARC) central processing unit 387 and a FIFO 394 for packet headers to be examined. The input to and output from the system is performed using 359k-gate 60 MHz MOM (MII [Media-Independent Interface] Octal MAC) ASICs 10 and 20 daisy-chained on the BlazeWire™ MAC Bus 60; the MOM chips 10 and 20 may each serve two Quad physical link chips (71 and 72, and 70 and 73, respectively) for local area Ethernets 63 or an interface for a wide area network such as the Distributed Access Device (DAD) WAN Processor 66 servicing T1 and POTS ("Plain Old Telephone Service") WAN lines 69 or for a Background Engine ("BE") 50.

FIG. 2 is a block diagram of the MOM chip, for example MOM chip 10, used in the preferred embodiment of the invention. Generally, the diagram shows MII interfaces 65 providing eight duplexed Ethernet ports. The receive interfaces 11 and parsers 12 receive the data packets, rewrite the frame headers as the

canonical headers described in Section 2 below, and divide the resulting packets into 128-byte cells, which are placed in FIFO 15 by producers 13 and the FIFO arbiter 14, in round robin arbitration among the eight ports. Data cells not bearing a canonical header (packet cells following the first cell of the packet) have a burst header added by burst logic 17 for internal tagging of the data. RX Credit Manager 19 adds transmission credits (discussed in Section 8 below) to the headers as appropriate to inform QM that the transmit FIFO 24 can accept more data to be transmitted. A token arbiter 18 determines when the data is to be sent to the MAC Bus TX cell 76 to be transmitted on the MAC bus 60 to QM 30.

Still referring to FIG. 2, data cells incoming on MAC Bus 60 are directed to the transmit consumers 26 and according to the circuit identifiers in their canonical/burst headers. The data packet headers are reconstructed and transmitted by the transmit consumers 26 and interfaces 27, and TX Credit Manager 28 is updated with credit information to be returned to the QM 30.

FIG. 3 is a block diagram of the QM 30 used in the preferred embodiment of the invention. Essentially, QM 30 is a collection of gates and state machines designed to rapidly execute the placing of data cells on appropriate queues (implemented virtually by linked lists of pointers to data buffers) and to apply queue policies as directed by the RE 40. The QM is divided into three clock regimes. The interface with the MAC bus 60 through Digital Delay Locked Loop 302 and Receive Interface 304, supplying Receive Data FIFO 306 and Receive Command FIFO 312, and through Digital Delay Locked Loop 301 and Transmit Interface 303 draining Transmit Data FIFO 305, are on the MAC bus clock. Data cells received are channeled directly through Dual RAMBUS® Access Cell 308 into the DRAMs 35 and 36 as discussed below. The DRAM Interface 307, operating on the DRAM clock, coordinates the operation of MAC bus FIFOs 305 and 306 as well as Header Out FIFO 309 (containing canonical header cells to be sent to the RE 40 [not shown] on Header Data Interface 74), Header In FIFO 310 (containing canonical headers rewritten at the Relay engine Data Interface 313 with appropriate routing

information) and DRAM Command FIFO 311. The latter contains the decisions of the RE as implemented by the QM logic shown in the network of functions under the SRAM clock domain. Receive Engine 315, Transmit Engine 316 and Header Prefetch Engine 324 direct the function of DRAM Arbiter 314 to provide instructions to the DRAM Interface 307 to move data in and out of DRAMs 35 and 36. Receive Engine 315 and Transmit Engine 316 also coordinate with Free Buffer Manager 318 to allocate buffers in DRAMs 35 and 36 to incoming data. EnQueue Manager 319 and DeQueue Manager 312 coordinate with Header Prefetch Engine and the Receive Queue State (head and tail of receive queue pointers, discussed in Section 5 below) 320, to determine, among other things, when cells containing canonical header data should be sent to the RE 40 for pattern-matching and the corresponding packets taken off the receive queue. The Header Prefetch engine 324 coordinates with the Relay Engine Context 326 and the Instruction Dispatch Manager 327, which receives instructions from RE 40 via interface 75, Relay Engine Instruction Interface 329 and the Relay Engine Instruction FIFO 328. Circuit Poller 317 polls the Transmit Engine 316 circuit by circuit to transmit cells and coordinates with the SRAM Arbiter 322, which through the SRAM Interface 323, accesses linked lists of buffer pointers ("descriptors") in SRAM 32 to track component cells of the packets as they are received and transmitted on one or more queues. These operations, where appropriate field mappings are hard-wired, provide for a great deal of flexibility in scheduling and routing executed at very high speed.

FIG. 4 is a block diagram of RE 40. A primary function of the RE 40 is to examine canonicalized packet headers received at interface 74 from the QM 30 and to determine rapidly whether the packet belongs to a known flow and to provide instructions accordingly on interface 75 for appropriate scheduling (quality of service). A CPU Core 387 (implemented with the ARC processor) contains an instruction cache 386 and a data cache 385 and communicates with the Code and Data DRAM 42 through the DRAM Interface 384 (which also accepts instructions from the BE 50 over a low speed bus 62 and the DMA 383 at initialization). String Compare Coprocessor 389 is used to aid the pattern recognition used to match a packet and flow. Generally, a

canonicalized packet header entering the RE 40 is pre-processed by the Hash Preprocessor 399 in parallel with being MUXed into Data FIFO 394 by MUXIn 394. The results of the parallel hashing are placed in Hash FIFO 393 and compared by the Hash Lookup Engine 392 with contents of the on-board L1 Cache of the Hash Table (of known hashes of header information associated with particular flow characteristics) 391. If no match is found in the L1 Cache 391, the Hash Lookup Engine will look to the entire Hash Table stored in Lookup SRAM 45, accessed through SRAM Interface and Arbitrator 388. Trie Search Coprocessor 390 is used to find the proper flow parameters in situations (discussed below) where the fast pattern matching is not appropriate or fails. With the flow parameters determined, an appropriate instruction is issued by CPU 387 into the Instruction FIFO 395 and processed by Instruction Push 396 multiplexed with any data from Data FIFO 394 by MUXOut 397 across interface 75 into the QM 30.

FIG. 5 shows a general schematic of the operation of the preferred embodiment of the invention from the queue management point of view. Data on MOM Receive Ports 15' are directed into the QM Main Receive FIFO 330. Also enqueued are data from WAN (T1 and POTS) port receive queues 69' processed under protocols 66' and under the direction of DAD Management 66" into a DAD Ethernet transmit queue 348' to appear on a MOM receive port 348. Data cells in the Receive FIFO 330 are placed in the main system packet memory DRAMs 35 and 36 while the canonical headers are forwarded in a FIFO 394 to the QM 30 where FastPath™ processes are applied to enable appropriate queuing of packets on per flow, per priority and per port queues 332 (static priority, as discussed below) and 333 (weighted robin priority, as discussed below) to be transmitted to the MOM Transmit Ports 24' (or the DAD 66 to be distributed on circuit queues 350 for further distribution to T1 and POTS Port Transmit Queues 69") for transmission. Fine tuning of scheduling may be achieved using Quality of Service Scheduling Process 336 relative to per flow queuing using Scheduled Queues 336 as "intermediate" queues. A Management Queue 337 is also provided with a Management Operations Process 338 operating on weighted round robin queues 333. A Monitor Queue 334 is also provided for

network monitoring information to be transmitted over MIIs 24'. On the BE 50 side, data placed on the MOM Port Transmit Queue 339 is transmitted over MII (100 Mbit Ethernet) link 64 into the BE Receive Queue 341. The Background Engine Main Forwarding Process 342 passes information into the BE Transmit Low Priority Queue 346 or the Management Queue 343 which is serviced by Management Operations Process 344 to develop data (including instructions) to be placed on BE Transmit High Priority Queue 345. Both BE Transmit Queues are drained into the MOM Port Receive Queue 347 via link 64 to be placed on the QM Receive Queue 330.

FIG. 6 is a generalized flow diagram for the process of the invention. It is to be understood that the processes occur simultaneously along various points in the diagram for different cells. Because the preferred embodiment of the invention divides often lengthy incoming Ethernet frames into cells for subsequent reassembly, it is important in the embodiment to characterize the cells relative to the packet from which it originated. A received cell may be a "start of packet" ("SOP") a "middle of packet" ("MOP"), an "end of packet" ("EOP"), or include a single packet as a "start and end of packet" ("SEP"). Because reception and transmission of data packets in the preferred embodiment is executed on a circuit-by-circuit basis, and a circuit is defined as a logical connection preserving the order of packets, cells of a packet on one circuit may be interleaved with cells of a packet on another circuit, for example on the MAC bus, but cells received on the same circuit must be transmitted in the same order. Thus, in FIG. 6A, with time going forward from top to bottom, an SOP 371 is received from Circuit 2, then an SEP 372 from Circuit 1, an SOP 373 from Circuit 3, an MOP 374 from Circuit 2, an EOP 376 from Circuit 3, an SOP 375 from Circuit 1 and an EOP 377 from Circuit 3, in order of appearance on the MAC bus.

Referring to the generalized process shown in FIG. 6, in operation 351, a packet is received at an MII and is split at operation 352 into cells by MOM 10 or 20 (referring to FIG. 1) which also adds canonical headers (and possibly burst headers). The cells in MOM Transmit buffers are arbitrated on the MAC bus in operation 353 and stored in DRAM for later transmission in

operation 354, which also includes the development of a procedure to associate the cells with the original packets, such as the link-list of virtual packets used in the preferred embodiment of the invention. If the cell is an SOP, a decision 355 is made to send the cell to a pattern matching procedure wherein the cell is hashed 356 and then matched 357 against known hash results associated with previously identified flows. If there is no match (possibly after several matching procedures), a new flow or exception is noted 358. In either case, an appropriate header is written 354 to appropriately schedule and route the packet. In the preferred embodiment, the scheduling is done by assignment of the packet to a queue associated with a specified quality of service and a particular circuit. A cell on a queue is transmitted 360 at the appropriate time, the process possibly including a rewriting of the headers. If the transmitted cell was an EOP, the packet is dequeued 361 from the circuit and if there are no other requirements for transmission of the packet (no more owners 362), the data buffer is released 363. This process may be further generalized and implemented in a diverse ways.

The flow of data through a preferred embodiment of the invention is presented below in further detail, which includes additional inventions.

2. Header "Canonicalization" and Frame "Cellularization"

Upon receiving a data packet on a physical link, the inventive network switch takes the Layers 2 and 3 headers of incoming packets (dropping any Layer 1 packet preamble) and converts it to canonical form. The invention further breaks variable-length packets into "cells" of a maximum convenient length for communication on the high-speed internal bus. This allows data packets of different lengths with different Layer 2 and 3 header formats, such as Ethernet "frames" or ATM "cells," to be routed by the same switching process and apparatus. The "canonicalization" of the header also aligns the header along 4-byte boundaries that are convenient for processing. The example here is for Ethernet frames, but is applicable to ATM cells with appropriate modification in the terminology and the interface ASIC.

Referring to FIG. 1, a frame of information is received by the MOM 1 chip 10 via one of the eight ports shown. The physical link Layer 1 processing is handled in the preferred embodiment by dual "off-the-shelf" Quad PHY integrated circuits (such as available from Lucent Technologies), each handling the transmission electronics of 10-Base-T (10 Mbit/sec) or 100-Base-TX (100 Mbit/sec) Ethernet. One of the ports, e.g., from MOM 2, may be connected by internal or external 10 Mbit Ethernet to a DAD integrated circuit including an off-the-shelf WAN processor (such as available from Motorola), which in turn interfaces with T1 and POTS lines via modem. Together, these form a QuadServe™ WAN access module.

Referring to FIG. 1, a frame or packet of information in the form of a data stream forming a message is input to a physical circuit 70 and then received by the MOM 1 chip 10 via one of its eight ports. FIG. 18 schematically illustrates the organization of a typical packet format. There may be a preamble 620, followed by a data link Layer 2 header 622, which contains information to bridge the packet, a network Layer 3 header 623, which contains information to route the message, and an application header 624, which contains information about the application for which the data is used. The headers are followed by the data itself 625, and, occasionally, there is a trailer 626, which usually is superfluous and not used.

The MOM 1 chip, preprogrammed in hardware in the preferred embodiment to recognize a variety of Ethernet protocols, drops the preamble and trailer, reads the Layers 2 and 3 headers from the received frame, and generates a canonical header of twenty-eight bytes, FIG. 7A. Having a buffer capacity of 256 bytes per port, the MOM 1 segments the frame data into cells of 128 bytes each (other cell lengths may be used in other embodiments).

Immediately adjoining the canonical header, Layer 3 (network) header information as received is stored. The Layer 3 header always starts at a multiple of four bytes from the start of the cell since the canonical header is 28 bytes. Important fields within the Layer 3 header are aligned at four-byte boundaries generally. This makes the processing of these fields very efficient for 32-bit processor/memory architectures.

Other header information from the higher layers, including the application layer, follow the Layer 3 header. The canonical header is placed at the beginning of the first cell of each frame or packet received and is used by the RE 40 to route or bridge the packet. When a packet is sent to the MOM for transmission, the MOM reconstructs the appropriate headers, preambles and trailers according to the destination and protocol information in the transmit canonical header and places the reconstructed packet on the line connected to the designated port.

FIG. 7A shows the organization and content of the canonical header in a preferred embodiment. The first two bytes 430 hold the circuit identification of the circuit on which the data packet was received, Byte 432, DL Info, provides information about the Data Link (Layer 2) header from the original received header. FIG. 7B shows the specific assignments to these bits. Bit 7 indicates if the received frame was VLAN (virtual local area network) tagged on reception. On transmission, if this bit is set, the outgoing packet is encapsulated with a VLAN header by the MOM chip handling the transmission. However, it should be noted that packets received with VLAN tags are not necessarily sent out with VLAN tags and vice-versa.

Bits 6 and 5 of FIG. 7B indicate how CRCs (cyclical redundancy checks) are to be handled. FIG. 7C is self-explanatory. Of note is that when the outgoing frame is different from the received frame, then a new CRC must be generated, but if the original frame is simply forwarded, then the CRC may not change, hence there is a need to retain the old CRC or generate another CRC. Bits 4 and 3 are unused and left as zeros. FIG. 7D shows the encoding for bits 2, 1, and 0 which identify the data link packet format.

The canonical header NL Info field 434 contains network layer information. FIG. 8A shows the meaning of the eight bits in the NL Info. Regarding reception: bit 7 true indicates that the destination address (DA) of the received information is the address of the bridge group associated with the circuit on which the packet was received; bit 6 true indicates that the DA is the system's address for the port; bit 5 true indicates that the DA is an address that has been pre-configured by the invention as a

"well-known address," such as one associated with a network control protocol. On transmission this bit is ignored. On transmission, if bits 7 and 6 are set, the appropriate source address is put on the SA field.

Bits 4-0 identify the Layer 3 protocol of the packet. FIG. 8B identifies those protocols preprogrammed into the invention. These can be extended as new protocols are developed and need to be handled efficiently by the system.

The Time Stamp four bytes 138 contain the time at which the packet will expire. The QM enters the time that the packet will expire when it receives the canonical header as part of the first cell of a packet. The QM upon transmitting a packet will check if the current time is greater than the time stamp value in the canonical header. If so, the data link device is directed to not transmit the packet and count it instead. When first generated by the MOM, this field contains cell information described in the "Data Flow In" section below.

The two-byte receive circuit identification (Rx Ckt Id) identifies the circuit on which the packet is received. The QM copies the receive circuit identification from the Ckt Id field 430 first supplied by MOM 1 before overwriting the Ckt Id field 430 with the circuit identification of the circuit on which the data is retransmitted. The receive circuit identification is thus retained for later use (such as for management and RMON functions by the BE 50).

DA is a 48-bit Layer 2 (MAC) destination address of the received packet.

SA is a 48-bit Layer 2 (MAC) source address of the received packet.

VLAN tag is a two-byte field to accommodate a packet received with an Ethernet 802.1Q tag. The VLAN tag bit in the DL Info field is also set, as described above. The MOM chip handling the transmission of this packet will tag the outgoing packet.

P-Type/len is a two-byte field containing the protocol type/length field. In this preferred embodiment, if the value is greater than 1500(decimal) this field represents a protocol, and if the value is less than or equal to 1500, this field represents a length. Protocol is captured in the Protocol Kind

subfield of the NL Info field. If the protocol is not so configured, the Protocol Kind subfield of the NL Info field would indicate Unknown (0) and the P-Type/len field would have the value. For example, if the packet was in the Ethernet 802.3 format, this field would contain the length which could be used for validity checks with length in the Layer 3 header.

The XX bytes may have other information based on the packet format of the received packet. FIG. 8C shows the contents of the XX bytes for the different DL format types.

3. BlazeWire™ High Speed MAC Bus

The received frame, reorganized into one or more cells, the first cell containing the canonical header and higher layer headers, is communicated to and from the QM on a high speed MAC bus called BlazeWire™.

The present design of BlazeWire™ is a full-duplex, clocked bus of ten signals and a clock signal each way between two large integrated circuit chips. The clocking protocol allows data transmission on the bus to be self-framing, asynchronous and non-aliasing. All the signals are differential signals between two conductor runs with the inherent transmission lines properly terminated. In this preferred embodiment, the electrical characteristics of the differential drivers and receivers are as substantially described in the low voltage differential standard (LVDS) ANSI/TIA/EIA-644. The differential signal voltage is about two hundred and fifty millivolts (250 mv), and the cable terminations and physical signal paths are arranged and designed to accommodate high speed operations over the bus. The bus is organized as a chain running from one large chip (MOM or QM) to another. A separate daisy chain token passing scheme is implemented as discussed below to control the access of the chips to the bus. The electronic design of the bus compensates for the practical variations inherent in different production runs of chips from possibly different manufacturers, supply voltage variations, and temperature variations. In preferred embodiments the speed of the bus can run upwards to the gigaHertz range.

The ten signals are composed of eight data, one parity, and one control. The data are placed on the lines on both the rising and falling edges of the clock signal. Since the data is placed on the signal lines at the clock transitions, the signals should be read at the receiving end at or very near the center of the clock signal. This allows any overshoots and any other signal delays or other anomalies to settle. Since the data is loaded onto the signal lines at both clock signal transitions, it is critical to have a symmetrical clock with minimum skew between the clock edges and the data being placed on the bus. The present circuitry provides a feedback mechanism for monitoring and finding the center of both phases of the clock signal, and furthermore to provide a symmetrical clock for the signals being sent out on the continuation of the bus through the chip.

FIG. 9 diagrammatically shows the basic signal flows between two sub-systems represented as MOM 1 and MOM 2 with twenty signal lines, a group of ten each way, and a clock with each group. FIG. 10 shows the differential character of each of the twenty-two lines. Differential drivers and receivers as known in the art are properly terminating the transmission lines in their characteristic impedance to maximize signal fidelity and minimize ringing. Other termination schemes such as on the drive side may be used to advantage in other embodiments.

FIG. 11 is a schematic of the circuitry by which one of the ten data bits is output from one of the MOMs. The circuitry is essentially duplicated for the other data bits. This circuit implementation maximizes clock symmetry and minimizes skew. The A data 462 is to be placed on the output 466 followed by the B data 464. The A data is latched in the flop 468 and presented to the logic array. Consider that the prior B data has remained in the latch 472 and is input to the logic array 460. The logic array is arranged to load a signal into the latch 474 which provides, when it is "exclusive or'ed" with the signal that remained in latch 476, the A signal at the output of the gate 466. On the next clock edge a similar operation provides the B data signal at the output, the B data 464 is latched 472 and "exclusive or'ed" with the prior signal in latch 474 such that the "exclusive or" of the data in latch 476 will provide the B

signal at the output of the "exclusive or" 466. FIG. 12 is a simplified timing diagram of the above.

FIG. 12A shows a composite timing chart of the bus clock and the ten data lines on the bus between MOMs 1 and 2. FIG. 12A shows the transferring of eight consecutive bytes (plus parity and control) on each edge of the clock signal.

When the signals are received at the MOM or QM, FIG. 13 shows the MOM's circuitry which is used to provide a delayed clock with an edge at the center of one phase of the received clock. Another similar circuit is used to provide a delayed clock with an edge at the center of the other phase of the received clock. These centered clocks are used to latch the data into the receive MOM and will be the basis for the symmetrical clock used to send out signals from the MOM. The received clock 80 becomes the data input to the latch 482 and latch 484. A delayed clock DLYA (a delay version of the input clock) latches the clock signal 480 into the latch 482 whose output is SAMPLE CLK A, and a delayed clock DLYB latches the clock signal 480 into the latch 484 with an output SAMPLE CLK B. The DLYA and DLYB are delayed by the control logic by a programmable amount. Both of these SAMPLE CLKs are fed back to a control logic array 90 through circuitry designed to synchronize the signals. In operation, the control logic can program when the DLYA occurs. In this way, the DLYA might latch the clock 480 signal when it is low which the control logic can determine by the SAMPLE CLK A signal. The control logic continues to set different delays until the clock 480 signal goes high. In a similar manner, the control logic continues to set different delays until the clock signal goes back low. As before, the control logic determines this condition from monitoring the SAMPLE CLK A signal. With reference to FIG. 13A, once the control logic has found the first rising edge 480' and the falling edge 480" of the clock signal 480, the control logic "knows" and can set the DLYA rising edge 486 at the center of the positive phase of the clock 480. This DLYA rising signal will be, effectively, the rising edge 486' used to latch data on the next successive positive phase of the clock 480. During the time that the centering of the DLYA signal, the actual data being received at the time 486, FIG. 13A, is latched by the

DLYB, FIG. 13, signal which had previously been centered to the positive phase of the clock 480. The previous centering of the DLYB was accomplished in the same manner as described above using the SAMPLE CLK B feedback signal and the DLYB delayed signal. In this embodiment, while one delayed clock is latching data, the other delayed clock is being centered for use at some later time.

The circuitry of FIG. 13 is duplicated to precisely measure the center of the negative phase of the input clock signal in order to latch in the data on the opposite phase. FIG. 13 shows the DLYC rising edge 489 precisely at the center of the negative phase of the received clock. As previously described, the DLYC clock is being centered during one negative phase of the clock 480 while the other (DLYD not shown) is latching data, and the DLYD will be centered while the DLYC clock latches data.

FIG. 14 shows parts of the delay circuitry. The IN signal 494 is delayed by one gate 495 and input to the "and" gate 496. If the control 1 signal is a logic one the signal traverses 496 and is output via the "or" structure 498 and becomes the output signal delayed by the three gate delays -- 495, 496, and 498. This delay is considered as a one unit delay. If the control 1 signal is a logic "0" and control 2 signal is a logic "1", the IN signal travels through gates 495 , 495', 496', 498' and 498. This path is longer by two gates, and the IN signal is considered to have gone through two single unit delay circuits. Each single delay unit adds two gate delays. If the control logic allows the IN signal to reach the three gates 500, and the control X signal is a logic one, the IN signal will go through an incremental of four gates -- the three gates 500 and the gate 504 (gate 502 being the common path duplicated in each delay circuit and disabled in prior delay circuits). This circuit adds four gate delays and forms a two unit delay. A four-unit delay (not shown) will replace the three gates 500 with seven gates, therefor adding an increment of eight gate delays or four unit delays. In this preferred embodiment, there are thirty-two single-unit delays, sixteen two-unit delays, and sixteen four-unit delays. The arrangement in this preferred embodiment allows an arithmetic-like progression of delays up to a total of 128 unit delays which may be selected. In other embodiments

other arrangements of delay circuits may be selected and other known delay circuits may be used to advantage. In this preferred embodiment, for expected manufacturing processes used to build the circuitry, and for expected temperature and supply voltage operation, a single unit delay will be about 0.15 nsec. It is expected that the variation of one unit delay may run from 0.08 to 0.3 nsec depending on the above mentioned parameters.

FIG. 15 (Table 1) is a table indicating the use of the control bit in this preferred embodiment. The bit is used for framing purposes. In the timing diagram of FIG. 12A, eight bytes are transferred on each clock transition marked by e0-e7. Table 1 shows the value of the control bit for the even numbers transitions, e0, e2, e4, and e6. The combinations indicate the allowable functions shown in the rightmost column. If the control bit is zero in each of the even transitions the bus is idling. Any of the combinations shown in rows 510 signal that the data on the data lines is a valid frame. In particular, since the value at the e6 time is always zero and the value at e0 time is always one for a valid frame of data, the system looks for a zero to one time sequence of the control bit. The one is assumed at e0, and if the combinations shown in rows 510 exists, the framing of the data shows a valid set of eight bytes.

The values of rows 510 are selected to ensure that no aliasing of valid frames of eight data bytes can occur. The valid control bit sequence combinations -- the rows 510, in FIG. 15 -- will always have a zero then a one, with no other zero/one patterns in a valid frame. FIG. 16 shows that the pattern of control bit values at the even clock transition shows frame 512 as invalid since there is another zero/one at e2 and e4 for that frame 512. However, the frame 514 is valid as is frame 516. In practice, the value of the control bit is measured at each receive clock phase and a zero to one transition separated by clock phase is monitored. When such a transition occurs, the one is treated as being in the e0 time slot and the monitoring of frame validity is based on that relative timing.

Transmission of data from the MOM chips to the QM is arbitrated by a token ring in the preferred embodiment. With reference back to the system block/schematic diagram FIG. 1, a

token ring arbitration path 61 is shown between MOM 1 and MOM 2. The token ring is a looped signal where a chip has the token when there is a logic difference between the incoming token signal and the outgoing token signal. In FIG. 17, there is no net inversion within the chips, so there is an inverter in the path so that at initialization one chip, in this case MOM 1, will be guaranteed to have the token and be in control of the bus. When a chip has the token it can send its own data over the bus, whereas when the chip does not have the token, it must wait for the token while other data are simply passed through the chip. When a chip has the token it will send out all the data needing to be sent by that chip before releasing the token. If MOM 1 has the token, it is passed to MOM 2 by MOM 1 changing the state of its output signal 61. MOM 2 then has the token.

4. Data Flow In

The MOM 1 chip 10 can store or buffer up to two cells or 256 bytes of received data for each of the eight ports. As described in the "Header Canonicalization" section above, the MOM chip reads the Layer 2 and 3 headers from the received frame or packet and generates an initial canonical header of twenty-eight bytes (described further in this section), followed by the network Layer 3 header and the application layer header in the first cell processed.

The MOM 10 (or 20) transmits the cell on the high-speed MAC bus 60 to the QM 30 when the MOM holds the token of the token ring arbitration path described above. Between the eight ports of a MOM, arbitration is round robin. The QM receives the cell and stores the cell in dynamic RAMs 35 and 36, in this preferred embodiment a RAMBUS® DRAM having two banks of DRAMs rapidly accessed as described in Section 9 below. Information describing a received, stored cell is placed in SRAM 32 and is called "descriptors." The canonical header is modified to include the Time Stamp. The modified canonical header and the rest of the header information in the first cell of the packet is placed in a Header Out FIFO 309 for transfer to the RE 40.

Because of the segmentation of frames, and the arbitration scheme, subsequent cells of a packet received on a circuit may be interleaved with cells of other packets received on other

circuits. To provide information to allow the QM to keep track of the order of the cells of a packet, the MOM writes an eight-byte (octbyte) "burst" header added to subsequent cells of the same packet (making up to 17 octbytes), corresponding to the first octbyte of the initial canonical header of the first cell of the packet.

Additional information is sent on the control signal line or bit of the high-speed MAC bus that allows identification of the boundaries of the cell and the type of information contained in the cell. FIG. 21 shows the use of the control bit to delineate data in groups of octbytes. The control bit 700 over eight consecutive clock phases frames eight bytes and distinguishes the data. The value of the control bit is shown as e0 through e7 in the table FIG. 22.

In FIG. 22 the even control bits, e0, e2, e4, and e6 are encoded as follows: e0 is always a one and e6 is always a zero to indicate that a valid group of eight bytes is received. To prevent aliasing of this encoding, the only values indicating a valid group are (for the even control bits, e0 through e6): 1000; 1100; and 1110. The bit e2 indicates the start of a cell, and e4 indicates the start of a packet. FIG. 23 shows a possible sequence of the even control bits: group 702 is not a valid group, while groups 704, 708 and 710 are valid. The circled zero/one 708 indicates that the only possible beginning to a valid group must have a zero followed directly by a one, and there can not be another zero/one in the next two bits (e2 and e4).

Still referring to FIG. 22, the odd control bits are encoded as follows: e1 indicates a transmission credit (see discussion below) exists, e3 (code bit 0) and e5 (code bit 1) form a two-bit end code, and e7 (short word) indicates an octbyte containing fewer than eight meaningful bytes. The short word can be used at the start of a cell or at the end of a cell.

FIG. 24 is a chart of several packet types that may be encountered. The first cell 720 of the packet may have up to sixteen octbytes, or 128 bytes. The even control bits 722 for the first 32-bit word (octbyte) is 1110. As shown in FIG. 22, this code means that this octbyte is part of a valid first cell of a packet. As shown, e0 equal to "1" is required for a valid

cell; e2 equal to "1" means this eight-byte transfer is the start of a cell, e4 equal to "1" means it is the start of a packet, and e6 must be zero for a valid cell. For the cell 720, the odd control bits are all zeros except for bit e5 of the last eight-byte transfer, which is a "1". FIG. 25 shows the encoding of the control bits e1, e3, e5, and e7 -- the odd control bits. For cell 720, e5 is a "1" and e3 is a "0" which decodes into "end of packet." Thus cell 720 is a one-cell packet (SEP). It should be noted that this cell need not be a full 128 bytes long.

Cell 724 is a valid starting cell of a packet, and here e3 of the odd control bits 726 is set meaning "end of cell" but not "end of packet"; thus, it is an SOP cell. The next cell 728 is the second cell of a packet (MOP), and all the cells following an SOP cell will have up to seventeen octbytes, including an octbyte burst header 330 added to the beginning of each cell. For this second cell, the last octbyte e3 is set meaning this cell is the end of a cell, but not the end of the packet. The cell 734 has e5 set in the last eight byte group, meaning that this cell is the end of the packet (EOP), and in this instance, e7 is also set. The bit e7 means that the last group of eight was not filled and was a "short word" (as so labeled in FIG. 25), and when this happens, the last byte 338 contains the number of valid bytes in the last eight byte group. For example, if there were only three valid bytes in the last group, the last byte (concurrent with the e7 control bit), would contain 0011, or decimal three.

Regarding the transmission of cells to the QM from the MOM chip, the first octbyte at the start of the first cell contains a portion of the canonical header that is modified by the QM to include the Time Stamp. The entire canonical header is stored in the DRAM with the other headers and such frame data as may fit in the remainder of the 128 bytes,

FIG. 26 shows the transformation of the first octbyte of the canonical header by the QM. As shown, the initial four bytes 740 written by the MOM, the Ckt Id, DL Info and NL Info, are carried forward by the QM. The second four bytes 742, including cell information, is overwritten by the QM with the Time Stamp 748. (The canonical header is sent to the RE, which

only deals with packet policy and is unconcerned with cell information.)

The first byte 744 of the cell information bytes 742 contains the number of transmission credits being reported from the QM (described in the "Transmission Credit Scheme" section below). The second byte contains credit flags, bit 7 being a SYNCH flag (for initialization) and bit 6 a "parent" flag (described in Section 8 below). The third byte provides cell information whose meanings are shown in FIG. 27. The bit meanings are: bit 7 indicates cell error; bit 6 packet time out; bit 5 a packet from the bad packet queue; bit 4 from the monitor queue; and bits 3-0 are selected bits from the control described above. Bit 3 is the packet end bit, bit 2 is the start of packet bit, bit 1 is the data cell bit, and bit zero is the transmit credit bit. The last byte in the cell information bytes 742 provides the cell length in number of bytes.

The octbyte-long burst header used to track cells without canonical headers is shown in FIG. 28. Its fields are identical to those of the first octbyte of the initial canonical header except that DL Info and NL Info (used by the RE which only sees the SOP) is replaced by the cell sequence number 752 and unused space. The Ckt Id 750 is used to match the cell (or more specifically, its proxy, the buffer descriptor) with preceding cells having the same Ckt Id, which should have sequential sequence numbers (unless a cell has been discarded). Once the cell is linked by the QM with preceding cells (as described below), the credits entered, and action taken on the other cell information, the burst header is no longer needed and is dropped. (A cell may be discarded if parity information detects an error. In such cases, at this time the cell and finally the packet is aborted by signaling the MOM chip.) A new burst header is created for the cell by the QM in the transmit phase, where the CKT ID shows where the packet is being sent.

5. QM Buffer and Queue Structure and Operation

Data cells received on the MAC bus by the QM are individually stored in the RAMBUS® DRAMs according to the fast-access operation described in Section 9 below, in addressable 128-byte data buffers, with the canonical header intact but

rewritten to include the Time Stamp, and with the burst header octbyte dropped. Address 00000 does not contain cell information and corresponds to a null-pointer.

All data cells received on the MAC bus and stored in data buffers are organized in a single virtual receive queue using a descriptor/pointer scheme that is used for all but a handful of specialized queues for exceptions. The scheme allows a receive queue corresponding to up to 1 Gbytes of data.

In the descriptor/pointer scheme, data buffer "descriptors" in the QM SRAM, comprising two 4-byte words, are surrogates for the actual data stored in the buffers and are linked to form logical packets. Thus a descriptor assigned to a data buffer with data has a field in the first word indicating the address of the buffer in the DRAM in which the associated cell is stored and a field in the second word containing a pointer to another descriptor 802 in the SRAM associated with the next cell of the same packet. As shown in FIG. 29, a complete multi-cell packet is described by a descriptor "link list," with the second word of the SOP buffer descriptor 801 pointing to the MOP buffer descriptor 802, the second word of descriptor 802 pointing to EOP buffer descriptor 803 and the second word of descriptor 803, associated with the last cell of the packet, containing a pointer pointing to descriptor 801, associated with the first cell of the packet. As shown in FIG. 29B, an incomplete packet has a null pointer in the second word of descriptor 805.

Queues are formed in the invention by a queue head pointer pointing to the first word of the descriptor associated with the first cell of the first packet in the queue and with a field in that first word pointing to the first word of the descriptor associated with the first cell of the next packet in the queue, and so linked reiteratively until the last packet in the queue, which has a queue tail pointer pointing to it, as shown in FIG. 30 with the receive queue head pointer pointing to the designator 812 associated with the first cell of the first packet in the queue and tail 811 pointing to the designator 815 associated with the first cell of the last packet of the receive queue (the descriptors each map to a 128-byte buffer in DRAMs 35 or 36). As shown, the queued packets are not necessarily complete, but in this packet-oriented implementation, data cells

received from the MAC bus are "added" to the packet to which it is identified by Rcv Ckt Id in the burst header, rather than at the end of the queue.

In the receive operation, the QM Descriptor SRAM is organized into a buffer descriptor table and a receive context (or circuit) table. The buffer table or list has descriptors containing two 4-byte words, with word 0 containing a buffer address of a data buffer in the RAMBUS® DRAM (hence the buffer table entry is an implicit buffer), and word 1 containing a pointer to another descriptor in the buffer table. At initialization, the buffer table is a "free buffer table" the designator of the first free buffer to which the QM hardware by a head pointer points and the second word of which points to the next free buffer descriptor, and so reiterated in a link until the last free buffer designator which contains a null terminator in its second word.

As a data cell is presented by the MAC bus to the QM, the QM extracts its circuit id from its canonical or burst header and checks for an entry in the receive context (circuit) table which yields information on the activity of that circuit. When an SOP is detected, an entry on the receive context table (8 bytes/circuit) is created and a pointer (current buffer) is entered pointing to the next free buffer designator. The cell data is written into the associated RAMBUS® DRAM buffer. The free buffer list pointer is moved to the next free buffer designator after the "current buffer" is allocated.

If the received cell was not an SEP, the second word in the buffer designator points to the next free buffer designator, preallocating the associated buffer, and a "0" is written in the second word of that next buffer entry.

If the received cell was an SEP or an EOP, the second word in the buffer descriptor is set to point to the first buffer descriptor for the packet, and the resulting link-list defining the packet is de-linked from the receive context table.

The cells received with the same circuit id, which may be interleaved on the MAC bus, are thus virtually reorganized by link-lists into packets, some of which may be incomplete even when leading cells are transmitted in cut-through operation. In

the latter case, as shown in FIG. 30B, the current buffer of the receive context table 820 points to the next buffer descriptor 833 corresponding to the buffer into which the data cell is to be loaded, and the buffer descriptor 833 is linked to the descriptors 832, 822, and 821 of the other cells of the packet, one of which, descriptor 832, is linked as the current buffer 821 of a circuit entry in the transmit context table. Since the circuit entry in the transmit context table provides routing information, the data subsequently placed in the buffer associated with descriptor 833 "knows where to go." This system of link management allows "cut-through," that is, the transmission of portions of a packet while other portions are still being received.

6. Relay Engine Processing/Flow Matching (FastPath™)

The receive queue of linked descriptors of SOPs waits for processing by the RE 40. The SOP cells themselves are loaded, as room is made available, into a "circular" FIFO 394 of 16 128-byte registers processed by the relay engine. This is implemented with a pointer system that follows the processing of the SOP cells, adding cells until the register is full (when the send pointer "catches up" to the receive pointer in FIG. 19), then adding another cell only when processing of the cell pointed to by a head pointer is complete and dropped (and the receive pointer "falls behind" the transmit pointer).

The RE operation centers around a four-stage pipeline. Pipelining is a term of art used for many years, especially in high speed hardware designs, and will not be further discussed herein except incidentally. The RE's task is to determine how to best forward a frame flow and to provide forwarding information accordingly to the QM to route and schedule retransmission of stored packets. The four stages are briefly described here, followed by a more detailed description of the hashing and signature functions used to perform pattern matching to identify a flow.

The first stage stores the full header information (the entire SOP cell) in a "circular" data FIFO, in parallel as the header is processed by a hash engine to compute a hash and a signature value to perform a pattern-matching function to check

whether the packet is part of an existing flow for which routing and scheduling information has already been developed.

The second stage receives the Hash value which is used to address a Hash Table L1 391. If a valid entry is found in this table, the signature from the L1 Table is compared to the computed signature of the Hashed data. If consistent, then a Flow Tag (not shown) from the Hash Table is presented to the next stage of the pipelined FE/RE hardware design together with an indication that a valid hit was found. The Flow Tag is a 24-bit index into a table in memory where information about the flow is stored. This information will include the circuit or circuits on which to forward the packet along with other flow related information as described elsewhere herein.

A valid Flow Tag pointer (linking the contents pointed to) is the preferred result of the pattern matching functions described in this preferred embodiment

If a match is not found in L1, the search is performed on the off-chip L2 Table 45. Signatures are compared as above and the Flow Tag from the L2 table is presented to the next stage. To facilitate the next search, the L2 entry is written into the L1 table.

If there is no hit in either L1 or L2, the computed hash and signature are presented to the next stage with an indication that no hit was found.

The third stage receives the above information and determines if the header look-up was successful. If successful, the header data is updated according to the protocol rules that apply and the packet is forwarded according to the flow information. However, if the header is found to be a TCP (Layer 4 Transport Control Protocol) SYN packet, or an equivalent start of connection packet in another protocol, or if the frame is not part of a known connection flow, the packet is not forwarded according to the flow information. In these instances the RE acts to route the frame by decoding the full pre-hashed header. In the process it creates useful flow information and inserts a tag that points to it in the L2 Hash Table using the hash and signature values obtained by the hardware in stage one.

In the fourth stage of the pipeline, the header is passed back to the QM to be queued for transmitting on the specified

queue according to the information supplied by the Flow Tag or the routing information supplied by the RE's decoding of the full pre- hashed header. For putting together the information to forward subsequent packets of the flow, the RE examines the application layer data in addition to the Layer 2 and Layer 3 headers.

In further detail, with reference to FIG. 4, when a packet is received, the QM 30 provides a useful header (as determined from the NL field) which may be as long as 128 bytes to the FE/RE by loading that header data onto a dual ported circular buffer in the RE. With reference to FIG. 4, the header data is sent from the QM 100 to the MUXIn 102 and placed on a FIFO stack DF in the RE 40. The RE uses the network link byte to index into a previously stored ordered data array of 128-bit entries, where each bit corresponds to one of the full received header data bytes. The bytes that correspond to the bits with a one are extracted and processed by the hash and signature functions. The byte string is padded at the end with zeroes to provide a string that is an even multiple of four bytes. In this preferred embodiment, up to 64 of the 128 header bytes can be processed by the hash/signature operation, but fewer or more can be used to advantage in other preferred embodiments.

The hash and the signature functions are identical except that different multipliers are used. But, in other preferred embodiments, other combinations of different multipliers and different divisors may be used to advantage.

With reference to FIG. 4, the Hash Preprocessor 399 inputs the selected bytes from the 128 bytes of the header data. The selected bytes form a number (n) of 32-bit words (multiples of 4 bytes, as noted above). The bits in this sequence of 32 bit words are treated as a polynomial in the Galois Field, GF[2] -- a Galois Field of 2 (Galois Field is known in the art). In this preferred embodiment, the polynomial is multiplied by a random 32-bit polynomial, and then divided by a carefully chosen polynomial of order 32 resulting in a 32-bit remainder. The divisor used above is selected to be both irreducible and primitive (irreducible and primitive are terms known in the art). A subset of the remainder bits are used as the actual index into the hash table. Bits 5 down to 0 are addresses

directed into the on-chip L1 cache 391. Bits 16 to 1 are used to address the 64K locations in the off-chip L2 RAM 45.

The divisor used in this preferred embodiment is $x^{32}+x^7+x^5+x^3+x^2+x+1$, although others may be used, provided they are both irreducible and primitive.

The contents of the Hash Tables which identify the Flow Tag and/or the destination of the incoming frame are organized as follows:

Hash Table 1 contains 64 words each of 64 bits, and it exists on chip to optimize the return of the value in the common occurrence where only a small number of flows are active. Larger tables can be used. In each word, see FIGS. 20A and 20B, bits 31-24 form a status where bit 31 being true indicates a valid entry. Bits 0-23 form a 24-bit Flow Tag where information about the particular flow is stored. The tag is a pointer to information about the circuit or circuits to which the packet will be forwarded. Obtaining the Flow Tag is the primary task of the RE. The Hash table also contains the 32-bit signature at bits 63-32, which is used to ensure that no collision has occurred and the result is valid. In order to further ensure the validity of the Flow Tag look up, the pre-hashed header data is stored so that unambiguous identification may be performed.

If there is no match in the L1 Hash table, the system will use the hashed result bits 16-0 to index into the 64k Hash Table L2. Each location will have a 64 bit width. Bit 30 is a Hash Bucket pointer wherein, if this bit is a zero, the bits in L2 table are organized functionally as in the L1 table. If there is one valid entry at this Hash Address, the system takes L2 bits 0-23 to be an index into a flow table to obtain a flow tag. See FIG. 20B. If there are no valid entries at this Hash Address, L2 bit 31, the Valid Bit, is set to a zero. If there are two or more entries at this hash address, then status word bit 30 is set to a one and the system takes the L2 bits 55-36 as a pointer to the Hash Bucket.

The Hash Bucket holds up to eight aliased addresses of 64-bit words. If the collision bit 29 is a one an aliased condition persists for both the hash and the signature operations and no further resolution will be performed by the hash mechanism, as no useful information can be obtained. At this point the two

conflicting flows are handed back to the processor to perform a Trie search for routing information. The eight words in the Hash Bucket are searched sequentially, and to facilitate this search the addresses are sequential starting at the lowest index into the table. If more than eight entries are directed to the Hash Bucket, the system reverts and the overflow are searched via the Trie routine. The Trie search uses a co-processor 390 and is organized as a large Trie database for routing and bridging.

The occurrence of signature and/or hash collisions can be monitored, and if excessive, the respective multipliers can be changed. Such changing results in a better randomization for the given set of addresses encountered in the network.

The hashing and signature routine results are not used in certain circumstances: when a connection is initiated, as when a TCP SYN or an equivalent "start of connection" packet arrives, or when a packet is found that does not belong to a connection flow, or the packet is part of a high security or other special mode. When such conditions are found the system can revert to the Trie search.

Generally processing of subsequent packets in a flow is accelerated by the optimization of software pattern matching as described above.

The RE returns information with instructions indicating which queue the cells are to be placed for forwarding along with the addressing. The QM receives the information and places the cells, which are stored in linked lists forming the contents of the packet which is being or was received, on a list to be transmitted.

7. Transmission Scheduling

The RE programs the QM, developing virtually by linked pointers in the QM Descriptor SRAM up to 16,000,000 transmit queues (24 bits) with managed priority for the various circuits. The core of the transmission phase is the Transmit Context Table, which is organized by circuit, four four-byte words for each circuit as shown in FIG. 35. Word 0 contains a credit sync bit, seven bits 812 for transmit credits (no transmission unless a credit exists for the circuit), a start of packet bit 814, and

23 bits designating the next buffer to transmit (next buffer ID). Word 1 816 contains eight flag bits 818. FIG. 35A shows the meaning of these flag bits: Bit 7 indicates that the packet is a single buffer; bit 6 indicates that the packet is bad, usually from a CRC error, and that the MOM should abort this packet; bit 5 indicates that the packet was dequeued from the monitor queue wherein the packet can be off loaded at some other port or to the background engine for traffic analysis; bit 4 indicates that the packet is "multi-owned" or may be transmitted to more than one circuit; bits 3-0 indicate the buffer length in bytes up to 128 bytes in groups of sixteen bytes. The remaining 24 bits of Word 1 contain the address of the first queue (each circuit may have 1, 2, 4, 8, or 16 associated queues). Word 2 820 in the transmit context table contains one bit 822 that indicates that a monitor queue is attached, four bits that indicate the queue service policy, and three bits that indicate a reference count. FIG. 35B shows the meanings of the four queue service policy bits. The possible designations are: one queue; two, four, eight or sixteen static queues; two, four, or eight weighted round robin queues; or two, four, eight and sixteen one-half static and one-half round robin queues. As described below, the static queues have the highest priority, followed by the weighted round robin queues. Word 3 contains the next circuit ID, and for "non-stand-by" circuits contains the parent circuit ID (discussed in the next section) and for "stand-by" circuits (discussed below) contains the stand-by scheduler control word.

The Queue Table shown at FIG. 36, which coordinates with the Transmit Context Table, contains four four-byte words for each queue. Word 0 contains a 2-byte standby circuit ID (discussed below) and two bytes of queue summary bits (only in every sixteenth queue number). Word 1 contains two bytes indicating the queue size and a 2-byte overflow counter ID. Word 2 contains a five-bit field indicating the number of standby queues and 24 bits for the head-of-queue pointer. Word 3 contains a 24-bit tail-of-queue pointer.

In the preferred embodiment, it should be remembered that a queue is formed by linking the SOP cells starting with a head-of-queue pointer to the first SOP (and a tail pointer to the

last SOP), and new cells of a packet are added to the cell of the packet. Thus, referring to FIG. 37, there are four SOPs in queue 16 of Queue Table 850, represented by linked descriptors 863, and two SOPs or "packets" in queue 17 represented by linked descriptors 864. Incomplete packets, such as that represented by linked descriptors 862 may nonetheless be transmitted (allowing "cut-through"), but transmission will stop on the circuit when the last descriptor indicates that its associated buffer is empty, thereby preserving the rule that packet order is preserved on a circuit.

The queue policy allows prioritizing and scheduling of transmission of data packets. Thus, under a fixed static priority, all the packets on a particular queue are transmitted before those on another. In a weighted round robin scheme, a certain number of packets on one queue are transmitted, then a certain number of packets on the next queue are transmitted, and so forth, this allows classes (queues) of traffic to have relative priorities without "starving" the lower priority classes. A "half-and-half" scheme is provided in which the static queues have priority, and when they are served.

A Schedule Table for the circuits in use is scanned continuously. As shown in FIG. 37, this is composed of a Primary Schedule Table with a Primary Schedule Table A 865 and a Primary Schedule Table B 866 and a Secondary Schedule Table 870. The Primary Schedule Table is located on-chip and consists of the two mentioned subtables, each with 64 entries. Slots in Primary Schedule Table A are visited once every Schedule Table time "tick." A Primary Table A entry contains a 6-bit index to an entry in Primary Schedule Table B. As shown in FIG. 37, any given Table B entry may have more than one Table A entry pointing to it. Primary Table B entries contain the size of the secondary table, and if the size is not equal to "0", then it also contains an offset into the secondary table 867 and the base address of the secondary table 868. If the size is equal to "0", the remaining fields are the "Use Parent Circuit" bit 871, the Parent Circuit ID 872 and the Circuit ID 873.

A cell transmission event is triggered when a schedule table entry with a Circuit ID is found. By entering the appropriate Circuit Ids in the Schedule Table, a cell

transmission ordering pattern is created which effectively allocates bandwidth to circuits according to their respective proportion of transmission events.

The hierarchical nature of the Schedule Table allows a wide range of rates to be programmed. This is done by "chaining" up to 3 levels of subtables. If the size field of a Primary Table B entry is not zero, this entry contains a pointer to a Secondary Table which is located off-chip. A Secondary Table 870 may have up to 255 entries, each of which may point to a Tertiary Table or may contain a Circuit ID. When table chaining is encountered, the offset field 867 is used to keep track of which entry is to be accessed in the lower-level table. At each visitation, the offset is incremented, modulo the table size.

The Stand-by Scheduler (SBS) is a secondary scheduling mechanism. As its name implies, it schedules traffic for bandwidth left over from the Schedule Table. There are 2 cases where stand-by traffic can be transmitted: (1) a transmit event resulted in no data sent for a circuit (lack of credits or lack of data); and (2) the Circuit ID programmed in the Schedule Table is zero, thereby pre-allocating a certain amount of bandwidth to stand-by traffic.

The SBS uses a version of the Calendar Queue algorithm, essentially a slotted time ring implemented as an array of linked lists. Each element of the array corresponds to a different time slot. Attached to each time slot is a list of circuits which are scheduled to send a packet at this time. A slot index advances with time. When a populated slot is found, a cell for the circuit at the head of the list at that slot can be transmitted. When a cell is transmitted for a particular circuit, the eligibility time for the next cell on that flow is calculated and mapped to another time slot.

Referring to FIG. 38, the Stand By Scheduler Calendar Table 878 is an on-chip table consisting of 64 entries. Each entry contains a head and tail index to described a linked list of circuits attached to a particular slot. The links are stored in the Next CCtId field of word 3 in the Transmit Context Table 860. The slot index 877 advances with periods corresponding to the QM core clock. When a SBS opportunity arises, the next circuit to transmit is found by scanning forward from the point

in time represented by the current value of the slot index. The next circuit to send is the one at the head of the list for the next populated slot. Once the next circuit is found, it is dequeued from the list and rescheduled. This field is a 6-bit number representing the number of Calendar Table slots between successive transmission events for the circuit. The next slot for a circuit is the current slot plus this interval, modulo the table size. The net effect of the SBS is an approximation of the Weighted Fair Queueing algorithm. The weight of a given circuit is the inverse of its SBS Interval.

Rescheduling is performed by calculating the next slot at which the circuit should be sent. The calculation of the next slot is based on the SBS Interval field of word 3 in the Transmit Context Table.

The "sending" in the preferred embodiment starts with the delinking of a packet string (which may be incomplete) from its queue ("dequeueing") and its linking to the current buffer of the Transmit Context Table 860 (as shown in FIG. 37). The circuit entries of the Transmit Context Table are then polled to send the buffer contents of the current buffer (if not empty) to the corresponding "circuit" 63'. Cell data is read from the RAMBUS® DRAMs according to the "ping-pong" scheme described below.

When a packet is fully transmitted, its buffers are returned to the free buffer list. Completion of transmission of a packet is indicated when the next buffer of the transmit context table is directed to the descriptor 880 associated with the first buffer of the packet by the second word of the descriptor 882 of the last buffer of the packet, referring to pointer 883 in FIG. 39A. The free buffer manager (not shown) then checks whether there are other "owners" (such as for multicasting) by looking at the "owner" field of descriptor 880 of the SOP, and if none (if value is one, otherwise decrement), as shown in FIG. 39B, it increments the free counter 890 by the buffer count 891 in the second word of descriptor 890. It moves the free buffer list head pointer 895 from the head of the free buffer list 896 to the descriptor to which descriptor 880 points, namely descriptor 881 of the buffer of the second cell, and enters in the next descriptor field of descriptor 880 a

pointer to the previous head of the free buffer list 896. As seen in FIG. 39B, all three buffers are thus linked at the head of the free buffer list.

8. Transmission Credit Loops

In the preferred embodiment, a hierarchical flow and congestion control scheme is provided by the use of two credit loops. A system of credits is established that indicates the ability of the MOM chip, for each of the eight output channels, to accept cells for transmission. As the MOM, for a particular channel is sending a packet, cell by cell, and as each cell is sent the MOM indicates, through the credit bits described above, that another cell can be transferred to the MOM chip. As shown in FIG. 31, the MOM, upon sending out a cell will increment the credit count 760, and as the QM transfers cells 762 to the MOM, the QM decrements the credit count 764. As noted above, the credits have a circuit ID such that the proper MOM channel credit is retained. In this preferred embodiment, as many as four transmit cells can be stored. The MOM has a FIFO in which the packet is reassembled from the cells.

When a cell is transmitted by the MOM chip, the credit sent back to the QM is a credit for a maximum length cell, which may be 17 octbytes when in cell mode or 16 octbytes when in packet mode (because the MOM deletes the burst header when in packet mode). However, the QM may send down something less than the maximum cell size. FIG. 32, which is duplicated for each output channel associated with the MOM chips, diagrammatically shows the mechanism by which the credits are processed in the MOM chip. There is a head pointer 770, a tail pointer 772, a virtual tail pointer 774, and a start of packet pointer 776. In this preferred embodiment there are 512, or four full 128-byte location in the transmit FIFO. In FIG. 32, there are 64 slots, each slot 778 representatively holding one octbyte. (The 64 octbytes equal the 512-byte storage capacity of the FIFO in this embodiment.)

At initialization the FIFO is empty, and the virtual tail is incremented, moving it through the FIFO locations. The virtual tail pointer stops when it reaches the head pointer. As the virtual tail pointer traverses a maximum cell length, a

single credit is sent via the transmit and receive credit mangers in the MOM chip. These credits are accumulated in the QM for this circuit. As the MOM receives cells to this circuit, the tail pointer (this pointer points to real information representing actual cell lengths) is incremented. If the QM sends less than a full cell, the virtual tail pointer is corrected. When the MOM actually transmits the cells the head pointer is incremented, and the virtual tail pointer is incremented. As the MOM sends out the cells the head pointer moves away from the virtual and the real tail pointers, opening up room in the FIFO. When the virtual tail pointer, which might have been corrected by the QM sending less than maximum cells, traverses a maximum cell length in the FIFO, as it catches up to the head pointer, a credit is sent and established in the QM.

The other remaining pointer, the start of packet pointer, 776, has one important function. That function is to retain the starting location of the start of the packet, so that if there is a collision on an Ethernet cable, the packet that was collided with can be retransmitted, in accordance with the published specification.

With regard to FIG. 2, the virtual tail and the real tail pointers are controlled by the transmit FIFO "producer," and the "consumer" controls the header and the start of packet pointers. But all the pointers are accessible to all the logic blocks within a MOM for comparison and for taking action.

FIG. 33 indicates how the MOM FIFO, a two-port, 64-octbyte memory, is controlled. An arbiter 780 controls the most significant three address bits of the FIFO from the "producer" side to keep track of the cells loaded from the QM, and the lower six bits, the total of nine bits needed to address the 512 locations, are controlled by the tail pointer 782 (one shown of eight). The virtual tail pointer 784 does not point to real data; it is a counter mechanism by which the credit manager can determine the number of credits to send to the QM. Another arbiter 786 and head pointers (one shown of eight) control the unloading and freeing up of the FIFO as packets are physically sent out by the MOM chip. The head pointer 788 controls the lower six bits of the FIFO from the unloading side of the FIFO. The consumer increments the head pointer as the data is sent

out. The head and start of header pointers are available to the transmit credit circuitry.

Referring to FIG. 26, a portion 742 of the first octbyte of the initial canonical header and, referring to FIG. 27, the burst header contain two credit flags, the "synch" flag and the "parent" flag. The synch flag is used at power up to properly establish the credit cycle operation described above. At power up, the MOM sends synch flags to the QM about every 10 milliseconds. When the QM has powered up, the QM looks for the synch flag, and when found the QM sends a synch acknowledge to the MOM. The MOM then will send up any credits as described above with the assurance that the QM is ready to accept the credits.

The parent flag is necessary since there can be a multiple of physical communication paths multiplexed into one channel of a MOM chip. When there is only one communication circuit connected to a MOM channel, as when the MOM is connected to an Ethernet, the credit system works as described above, but with many separate paths into one MOM channel, a method of maintaining credits for each of the paths connected to the one MOM channel was designed. One important aspect of this credit system is that it was necessary to ensure that none of the several communications paths connected to the one MOM channel could be blocked or locked out by another of the communication paths. In this embodiment, FIG. 34 shows two FIFO channels in a MOM chip. FIFO 800 operates with a single communications path. In this case, the MOM FIFO 800 is termed a leaf to indicate its operation with a single communications circuit. But FIFO 802 is associated with a FIFO channel that is connected to another chip, for example, a DAD chip 804 in this preferred embodiment, where the DAD is further connected to eight other communication circuits 804. In this case the FIFO 802 is termed a "parent" and the eight communications circuits connected to the DAD are the leaves. In this circumstance the QM maintains a credit for the individual leaves attached to the parent FIFO in the MOM. In this way the QM knows when the transmit FIFOs are filled and can accept no further cells. The QM can subsequently transfer cells to the other leaf by simply polling the credits in the parent and the leaves and transmit cells accordingly. In this

manner one leaf cannot prevent the servicing of the other leaves.

Referring to FIG. 38, in the Schedule Table 866 in the QM, there is an indication 871 whether there is a parent associated with that particular circuit. The MOM, acting as a parent, sends up credits for the parent FIFO and for each of the leaves associated with that parent.

The Parent Credit Table 875 is a 64-entry on-chip table in the QM. Each entry contains a credit count for what is treated as a "parent circuit." When a circuit is bound to a parent circuit, it can only transmit cells onto the MAC bus if it has credits available in both its Transmit Context Table credit field *and* in its parents credit field in the Parent Credit Table.

When a cell is transmitted for a circuit with a parent, both the Transmit Context Table credits and the associated parent credits are decremented. Parent credit update cells from the parent channels are sent back to the QM which causes the parent credits to be incremented.

The Schedule Table is used to bind a circuit to a given parent circuit. The Use Parent Circuit Bit (P) 871 and the Parent Circuit ID field 872 are used for this purpose. If the schedule table entry has the P bit set, this means that this circuit has a parent and should use the Parent Circuit ID 872 to index the Parent Credit Table 875.

9. Ultra-High Speed Access on RAMBUS®

RAMBUS® DRAMs 35 and 36 are off-the-shelf items. In the present invention they are used in a unique manner that maximizes the reading and writing bandwidth of the RAMBUS® for this data communication application.

The invention provides an interface 308 to the RAMBUS® which utilizes the dual bank organization of a RAMBUS® to increase the useful bandwidth of the RAMBUS® memory. Dual FIFO stacks are used with a controller to alternately address the separate DRAM banks within the RAMBUS®. The FIFOs increase the latency and increase the hardware overhead of the RAMBUS® controlling electronics, but attempts to guarantee that the sequential data written or read comes from the alternate banks.

In this manner, one bank is precharging while the other is being accessed, and then the other bank is precharging while the first bank is accessed.

Referring to FIG. 40, a RAMBUS® 900, is shown in block form showing the phase-locked loop, PLL, and the two dynamic RAM banks DRAM 1 and 2 (36, 37 respectively). The multiplexed data/address bus into and out of the RAMBUS® is essentially an eight-bit wide serial port with an accompanying clock.

The organization of data buffers in DRAMs 35 and 36 is such that all even data buffers (of 128 bytes) are on one bank and all odd data buffers are on the other. The arbiter 902 determines the order in which various requests for data are loaded onto FIFO stacks 904 and 906. The buffer addresses in the requests are either even or odd, and the requests with even buffers are loaded into FIFO 904 and the odd buffers into FIFO 906.

In the condition that the FIFOs are empty, the requests are loaded into the even or odd FIFO and the interleaver 908 transfers the request to the controller 910. However, as the requests become numerous, the requests in the FIFOs back up. When the requests have backed up into both FIFOs, the interleaver 908 takes the requests alternately from one FIFO and then the other ("ping-ponging"). Since these buffer addresses are alternately even and then odd, the controller accesses the two different banks in the RAMBUS® in an alternate or interleaved manner. In this operation, the first bank is being accessed while the second bank is being precharged, and, on the next access, the second bank will be accessed while the first bank is being precharged.

This alternative accessing substantially provides the fastest accessing for either writing or reading of the RAMBUS® and maximizes the throughput of the RAMBUS® memory as long as there are requests in both FIFO stacks, which is likely in high traffic situations. In contrast, requests presented on a purely FIFO basis likely will have a fractional number with back-to-back even or back-to-back odd requests causing a fractional number of time-outs to allow precharging.

Any latency relative to a particular request may in any case have occurred under normal access methods. The method here assures maximum usage of RAMBUS® resources under high traffic conditions.

10. Background Engine/Initialization

An important part of the invention is the use of the BE, interfaced on a MOM port during operation to perform monitoring and other higher-layer decision making. This allows for the BlazeWatch™ and Learn-and-Lock™ security systems for access to configuration and control functions, among other applications.

With reference to FIG. 1, a Boot FLASH ROM 51 is provided that is accessible to BE 50 for initialization and start up of the system. The boot ROM instructions will run when there is a power up or a complete system reset. The boot will test and verify that the section of the BE DRAM 53 is operational and reliable. This section is where the ISB code and the BlazeNet Runtime Kernel (BeRT) will reside. The first 1F (hex) or 32 (decimal) addresses of ROM 51 hold the initial interrupt vectors. Addresses 20-7F hold ROM information; 80-FF hold console support interface Routines, 100-4FF hold a MOM attribute table; 500-1FFFB hold the boot image; and 1FFFC-1FFFF hold the boot image checksum of a cyclical redundancy check (CRC). In this embodiment, the remaining BE DRAM 53 will be tested in parallel with running the BeRT initialization process.

The boot also tests the interrupt structure and operation to insure that the BARK (the background engine kernel) can receive interrupts, for example, from timers. Next the boot will initialize the I2C bus 62 and assign addresses to the chips attached to the I2C bus. The boot then determines the ID of chips on the bus, including revision level. The boot then looks up the ID of the chips found, and an initializer is found in the boot directory which is downloaded and executed.

The main system image is in the NonVolatile Storage 52 in a compact flash card containing, for example 10 Mbytes of system software. Basic information is transferred on the I2C bus to the RE 40 and MOMs 10 and 20. The complete image is subsequently transferred on the DMA channel 64.

The above discussion describes the preferred embodiment of the invention(s) at the time of filing. It should be clear that equivalent components and functions may be substituted without departing from the substance of the invention(s). Various mixes of hardware and software implementation are possible while retaining the benefits of the invention(s). Because the invention is intended to be highly flexible and scalable, it is the cooperation of the modules here disclosed that is important, rather than the number of modules and ports.

CLAIMS

What is claimed is:

1. A method for creating, identifying, retrieving and sending the switching flow path switching information of a packet of information bytes comprising the steps of:

retrieving the header of the packet;

selecting a programmable subset of said header bytes;

hashing the subset to form a hash result;

second hashing of the header to form a signature result;

selecting portions of the hash result to form an index into a memory;

determining that there is no valid flow tag at the index

decoding the header of the packet and determining the forwarding flow path;

creating a flow tag pointer for this packet;

loading the flow tag pointer at the index;

loading the flow path switching information at said flow tag pointer; and

sending the flow path switching information for forwarding the packet.

2. The method of claim 1 further comprising the steps of:

receiving a subsequent packet having similar relevant header bytes as the previous packet in the flow path;

hashing the header to form a hash result and a signature result;

retrieving a valid flow tag pointer at the index;

checking the newly calculated signature result with the previously calculated signature result to verify a valid flow tag pointer; and

retrieving the flow path switching information from the flow tag pointer location.

3. The method of claim 1 further comprising the step of:

selecting a programmable number of bytes at programmable positions from the header;

forming an ordered set from said selected number of bytes;

multiplying said ordered set by a random polynomial to form a product polynomial;

forming an irreducible and primitive polynomial; and

dividing said product polynomial by said irreducible and primitive polynomial forming a remainder polynomial,

wherein said remainder is the hash result.

4. The method of claim 3 further comprising the steps of:

multiplying said order set by a second random polynomial to form a second product polynomial,

dividing said second product polynomial by said irreducible and primitive polynomial forming a remainder polynomial to form a signature result,

storing the signature result,

wherein comparing said stored signature result to a signature result from a second verifies that the flow tag pointer is valid.

5. The method of claim 1 further comprising the steps of:

retrieving several additional packet headers in sequence; and

processing the additional packet headers by

pipelining the steps of claim 1 as applied to all said headers of the packets sequentially, wherein the processing steps of the packets overlap each other but are offset in time.

6. A method for identifying, retrieving and sending the flow path switching information of a packet of information comprising the steps of:

retrieving the header of the packet;

selecting relevant bytes in the header;

forming an ordered set of said selected bytes;

hashing the ordered set first to form a hash result and, with changed hashing parameters to form a signature result;

selecting portions of the hash result to form an index into a memory;

retrieving a valid flow tag pointer from the memory contents at the index;

verifying the validity of the flow tag pointer by comparing the calculated signature result with a previously stored signature result;

retrieving switching information stored at the flow tag pointer;

sending the flow switching information for forwarding the packet; and

updating the flow switching information of the packet header.

7. The method as defined in claim 6 wherein the hashing step comprises the steps of:

multiplying the header with a polynomial to form a first result;

dividing the result by another polynomial to form the hash result;

multiplying said order set by a second random polynomial forming a second product polynomial;

dividing said second product polynomial by said irreducible and primitive polynomial forming a remainder polynomial to form the signature result; and

comparing stored signature results with presently calculated signatures results to determine that said flow tag pointer is valid.

8. Apparatus for determining the flow path switching information of a packet comprising:

means for retrieving the header of the packet;

means for hashing the header to form a hash result;

means for second hashing of the header to form a signature result;

means for selecting portions of the hash result to form an index into a memory;

means for determining that there is no valid flow tag at the index;

means for decoding the header of the packet and determining the forwarding flow path;

means for creating a flow tag pointer for this packet;

means for loading the flow tag pointer at the index;

means for loading the flow path forwarding information at said flow tag pointer; and

means for sending the forwarding flow path information for forwarding the packet.

9. The apparatus of claim 8 further comprising:

means for receiving a second packet having substantially the same header as the first packet;

means for hashing the header to form a hash result and a signature result;

means for retrieving a valid flow tag pointer at the index;

means for checking the newly calculated signature result with the previously calculated signature result to verify a valid flow tag pointer; and

means for retrieving the forwarding flow path information from the flow tag pointer location.

10. A method for forming an index into an information field from an incoming byte stream comprising the steps of:

selecting a programmable number of said bytes at programmable byte positions;

forming an ordered set from said number of bytes;

multiplying said ordered set by a random polynomial forming a product polynomial;

forming an irreducible and primitive polynomial;

dividing said product polynomial by said irreducible and primitive polynomial forming a remainder polynomial, referred to as a hash result; and

forming said index by selecting portions of said hash result.

11. The method as defined in claim 10 for forming an index further comprising the steps of:

multiplying said ordered set by a second random polynomial forming a second product polynomial;

dividing said second product polynomial by said irreducible and primitive polynomial forming a remainder polynomial to form a signature result; and

comparing said signature results to a signature previously stored in the index to determine that said index is valid.

12. The method as defined in claim 11 wherein the step of forming the signature result includes calculations involving lossy compressions including CRC's and checksums, and

storing the signature result as part of the hash result.

13. The method of claim 11 further comprising the step of using the signature result as a hash key directly to increase the confidence of a non-aliased hash result.

14. The method of claim 11 further comprising the step of using the signature result to form a perfect hash result with no aliasing within the known transport layer flow.

15. In a process for switching flow of sequential generic data packets according to forwarding rules applicable to a particular flow, the process of determining whether a presently received data packet is part of a flow initiated by a previously received data packet, said process comprising:

(a) hashing a portion of said presently received data packet; and

(b) comparing a portion of said hash result with a corresponding portion of said previously received data packet.

16. The process of Claim 15 wherein the step (b) of comparing the hash result is performed indirectly by selecting corresponding portions of the respective hash results as an index to a memory location, that includes information that is checked to determine whether said presently received packet is feasibly part of said flow.

17. The process of claim 16 wherein the information checked includes the hash result of a larger portion of said previously received data packet, which is compared to the hash result of a corresponding portion of said presently received data packet.

18. The process of Claim 15 further comprising converting said presently received data packet into one or more canonical cells, corresponding portions of each of which are hashed in step (a) of said process, and corresponding portions of each hash result compared to a corresponding portion of said previously received data packet.

19. Apparatus for determining whether a presently received data packet is part of a flow initiated by a previously received data packet, said apparatus comprising:

(a) means for hashing a portion of said presently received data packet;

(b) means for comparing a portion of said hash result with a corresponding portion of said previously received data packet.

20. The apparatus of Claim 19 further comprising means for converting said presently received data packet into one or more canonical cells, corresponding portions of each of which are hashed by said hashing means and corresponding portions of each

hash result compared by said comparison means to corresponding portion of said previously received data packet.

21. Apparatus for determining whether a presently received data packet is part of a flow initiated by a previously received data packet, said apparatus comprising:

(a) a hash preprocessor having an input of a portion of said presently received data packet and an output of a hash result; and

(b) a comparator having inputs of said hash result and a corresponding hash result of said previously received data packet, and having an output tagging said presently received data packet as part of said flow if and only if there is a match.

22. The apparatus of Claim 21 wherein said hash preprocessor (i) selects a programmable number of bytes of said portion of said data packet at programmable byte positions, (ii) forms an ordered set from said number of bytes, (iii) multiplies said ordered set by a random polynomial to form a product polynomial, (iv) forms an irreducible and primitive polynomial, (v) divides said product polynomial by said irreducible and primitive polynomial to form a remainder polynomial to produce said hash result.

23. The apparatus of Claim 21 wherein said comparator comprises (i) a memory with a plurality of indexed locations exactly one of which includes information about said flow initiated by said previously received data packet, (ii) a pointer to said indexed locations set by a portion of said hash result, and (iii) a comparator with inputs of at least a portion of said information about said flow and of corresponding information of said presently received data packet and an output determining said tagging.

24. The apparatus of Claim 21 further comprising a second hash processor having an input of a larger portion of said presently received data packet and an output of a signature result, which is compared to the corresponding signature result of a corresponding portion of said presently received data packet.

25. The apparatus of Claim 24 wherein said second hash preprocessor (i) selects a programmable number of bytes of a portion of said data packet at programmable byte positions, (ii)

forms an ordered set from said number of bytes, (iii) multiplies said ordered set by a random polynomial to form a product polynomial, (iv) forms an irreducible and primitive polynomial, and (v) divides said product polynomial by said irreducible and primitive polynomial to form a remainder polynomial to produce said signature result.

26. The apparatus of Claim 22 further comprising a second hash preprocessor having an input of a larger portion of said presently received data packet and an output of a signature result, which is compared to the corresponding signature result of a corresponding portion of said presently received data packet wherein said second hash preprocessor (vi) multiplies said ordered set by a second random polynomial to form a second product polynomial and (vii) divides said second product polynomial by said irreducible and primitive polynomial to form a remainder polynomial to produce said signature result.

27. The apparatus of Claim 26 wherein said comparator comprises: (a1) a memory with a plurality of indexed locations exactly one of which includes said signature result of said previously received data packet, (a2) a pointer to said indexed locations set by a portion of said hash result, and (a3) a comparator with inputs including (i) said signature result of said presently received data packet and (ii) corresponding information contained in that of said memory locations pointed to by said pointer; and an output determining said tagging only if there is a match of said inputs.

28. The apparatus of Claim 21 further comprising one or more input memory registers defining said presently and previously received data packets as components of potentially larger data packets received by said apparatus as part of an input data flow.

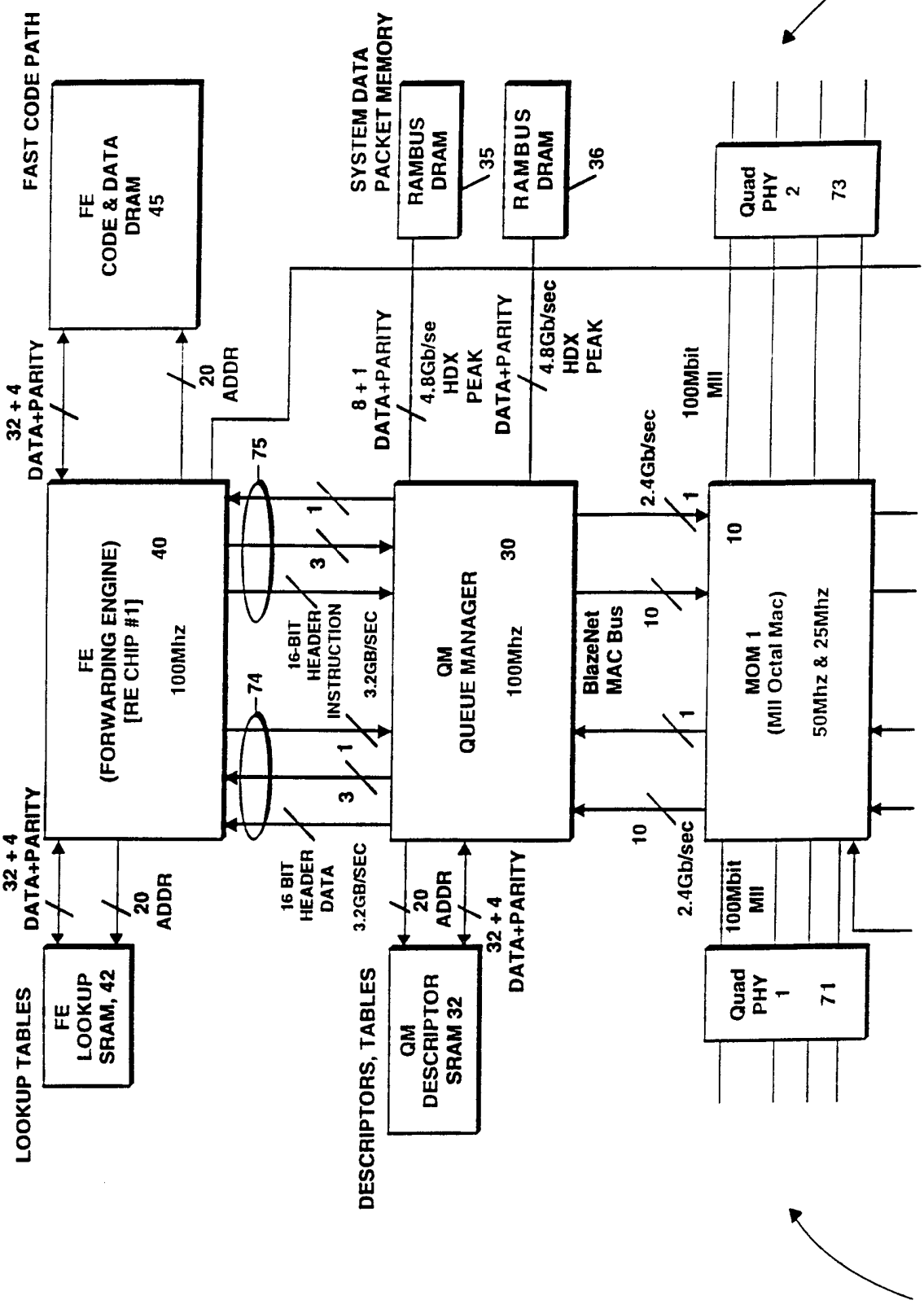

FAST CODE PATH
FE
CODE & DATA
DRAM
45
SYSTEM DATA
PACKET MEMORY
RAMBUS
DRAM
35
RAMBUS
DRAM
36
Quad
PHY
2
73
32 + 4
DATA+PARITY
20
ADDR
75
8 + 1
DATA+PARITY
4.8Gb/se
HDX
PEAK
DATA+PARITY
4.8Gb/sec
HDX
PEAK
100Mbit
MII
2.4Gb/sec
FE
(FORWARDING ENGINE)
[RE CHIP #1]
40
100Mhz
1
3
16-BIT
HEADER/
INSTRUCTION
3.2GB/SEC
74
1
3
QM
QUEUE MANAGER
30
100Mhz
BlazeNet
MAC Bus
10
1
10
MOM 1
(MII Octal Mac)
50Mhz & 25Mhz
32 + 4
DATA+PARITY
20
ADDR
16 BIT
HEADER
DATA
3.2GB/SEC
20
ADDR
32 + 4
DATA+PARITY
10
2.4Gb/sec
100Mbit
MII
LOOKUP TABLES
FE
LOOKUP
SRAM, 42
DESCRIPTORS, TABLES
QM
DESCRIPTOR
SRAM 32
Quad
PHY
1
71

Figure 1

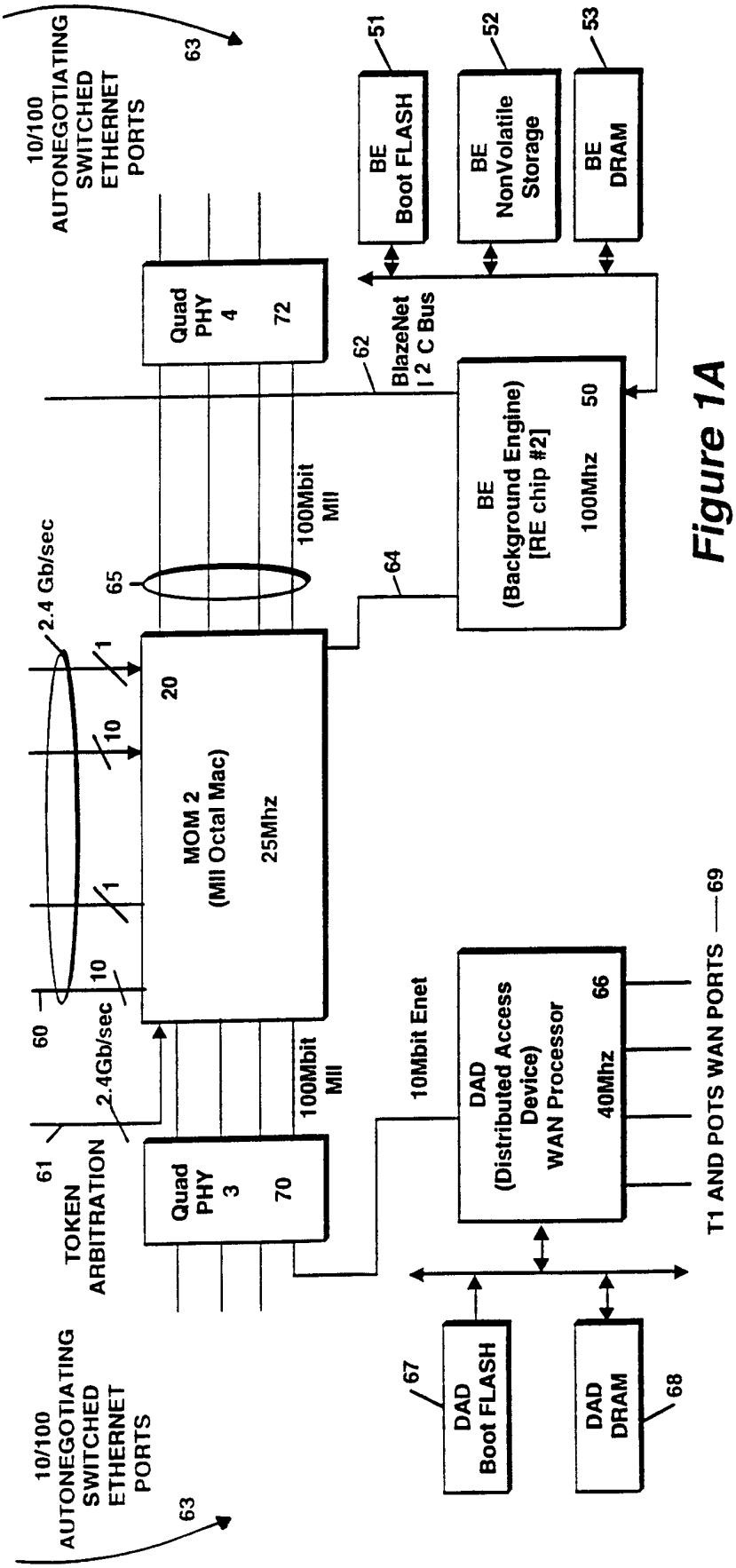

10/100
AUTONEGOTIATING
SWITCHED
ETHERNET
PORTS
63
BE
Boot FLASH
51
BE
NonVolatile
Storage
52
BE
DRAM
53
Quad
PHY
4
72
62
BlazeNet
I 2 C Bus
BE
(Background Engine)
[RE chip #2]
100Mhz
50
100Mbit
MII
65
64
2.4 Gb/sec
1
10
20
MOM 2
(MII Octal Mac)
25Mhz
1
10
60
2.4Gb/sec
61
TOKEN
ARBITRATION
Quad
PHY
3
70
100Mbit
MII
10Mbit Enet
DAD
(Distributed Access
Device)
WAN Processor
40Mhz
66
T1 AND POTS WAN PORTS
69
10/100
AUTONEGOTIATING
SWITCHED
ETHERNET
PORTS
63
67
DAD
Boot FLASH
DAD
DRAM
68

Figure 1A

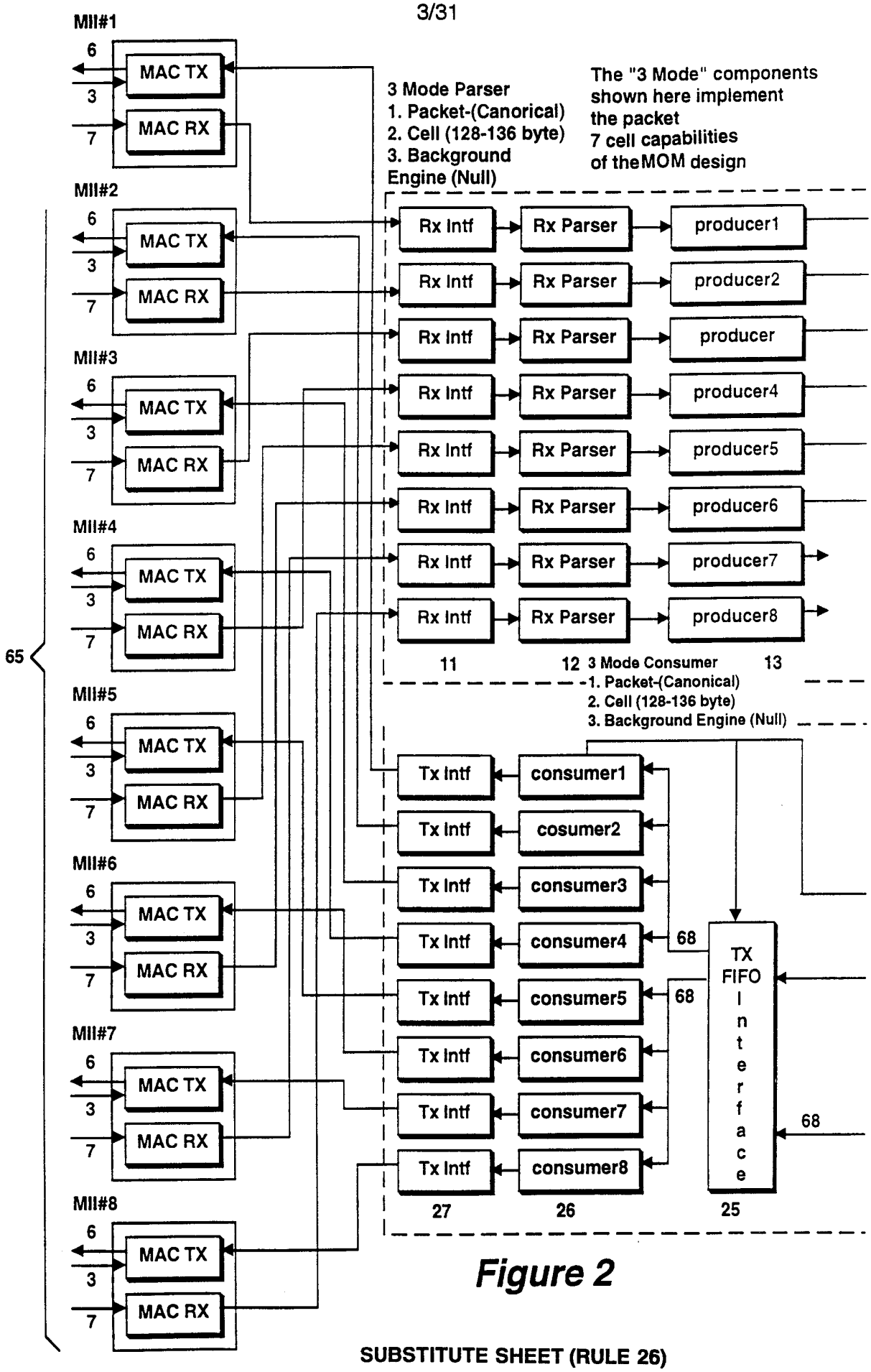

MII#1
MII#2
MII#3
MII#4
MII#5
MII#6
MII#7
MII#8
6
3
7
MAC TX
MAC RX
65
3 Mode Parser
1. Packet-(Canorical)
2. Cell (128-136 byte)
3. Background
Engine (Null)
The "3 Mode" components
shown here implement
the packet
7 cell capabilities
of theMOM design
Rx Intf
Rx Parser
producer1
producer2
producer
producer4
producer5
producer6
producer7
producer8
11
12
3 Mode Consumer
1. Packet-(Canonical)
2. Cell (128-136 byte)
3. Background Engine (Null)
13
Tx Intf
consumer1
cosumer2
consumer3
consumer4
consumer5
consumer6
consumer7
consumer8
68
68
68
TX
FIFO
Interface
27
26
25

Figure 2

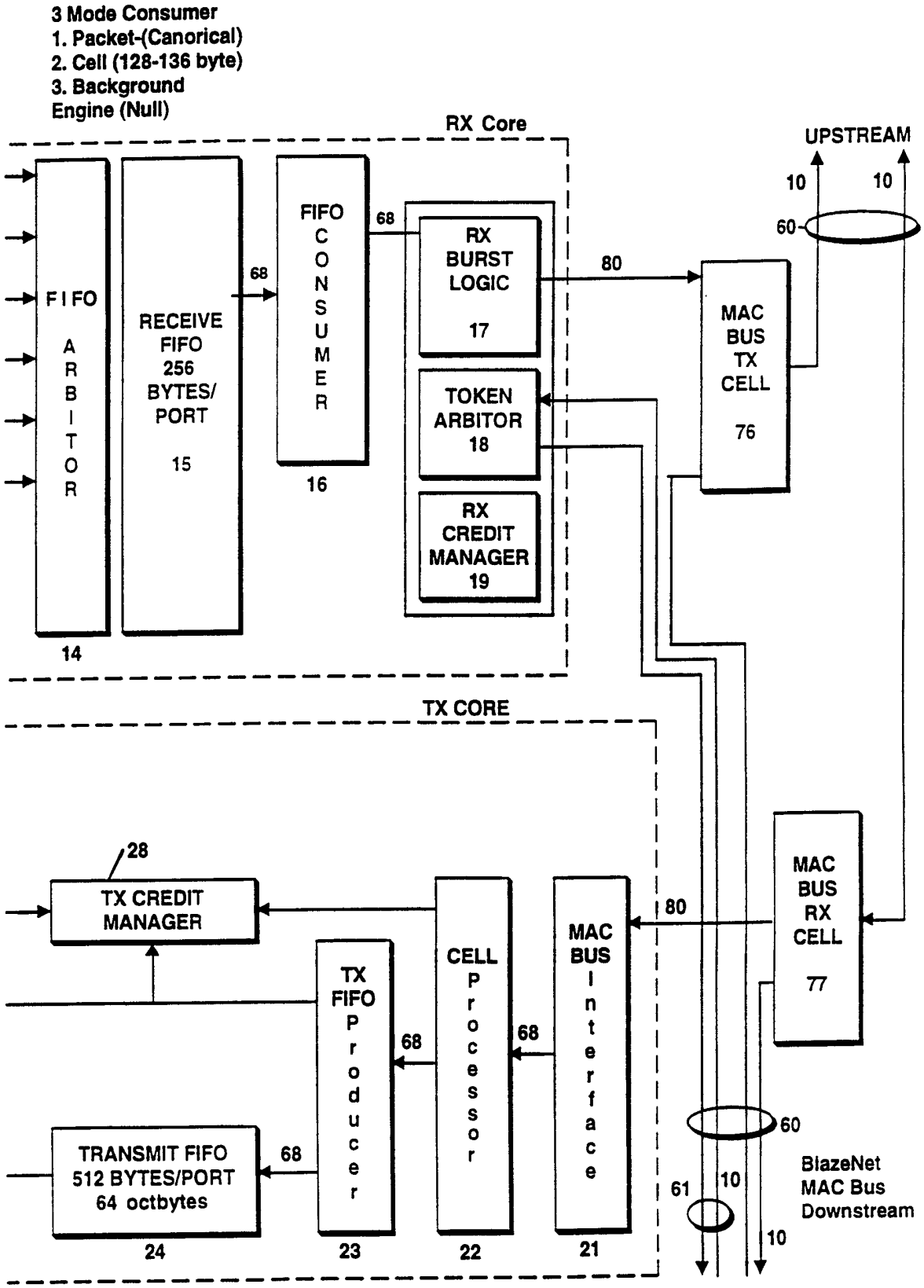

3 Mode Consumer
1. Packet-(Canorical)
2. Cell (128-136 byte)
3. Background
Engine (Null)
RX Core
UPSTREAM
10
10
60
FIFO ARBITOR
14
RECEIVE FIFO 256 BYTES/ PORT
15
68
FIFO CONSUMER
16
68
RX BURST LOGIC
17
80
TOKEN ARBITOR
18
RX CREDIT MANAGER
19
MAC BUS TX CELL
76
TX CORE
28
TX CREDIT MANAGER
TX FIFO Producer
23
68
CELL Processor
22
68
MAC BUS Interface
21
80
MAC BUS RX CELL
77
60
61
10
10
BlazeNet MAC Bus Downstream
TRANSMIT FIFO 512 BYTES/PORT 64 octbytes
24
68

Figure 2A

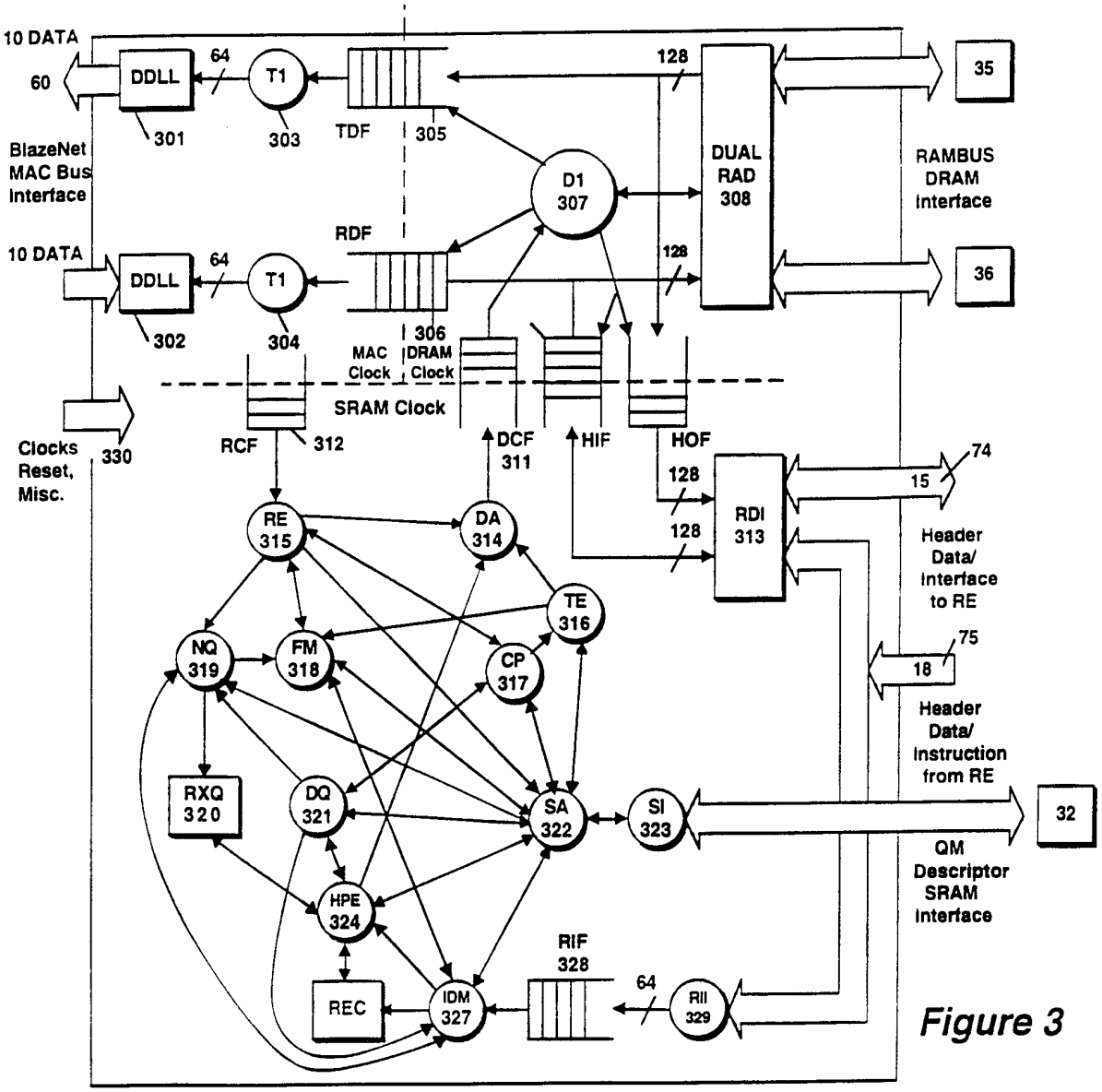

10 DATA
60
DDLL
64
T1
301
303
TDF
305
128
35
BlazeNet MAC Bus Interface
D1 307
DUAL RAD 308
RAMBUS DRAM Interface
RDF
10 DATA
DDLL
64
T1
128
36
302
304
306
MAC Clock
DRAM Clock
SRAM Clock
Clocks Reset, Misc.
330
RCF
312
DCF 311
HIF
HOF
128
74
15
RDI 313
128
Header Data/ Interface to RE
RE 315
DA 314
TE 316
75
18
NQ 319
FM 318
CP 317
Header Data/ Instruction from RE
RXQ 320
DQ 321
SA 322
SI 323
32
QM Descriptor SRAM Interface
HPE 324
RIF 328
64
REC
IDM 327
RII 329

Figure 3

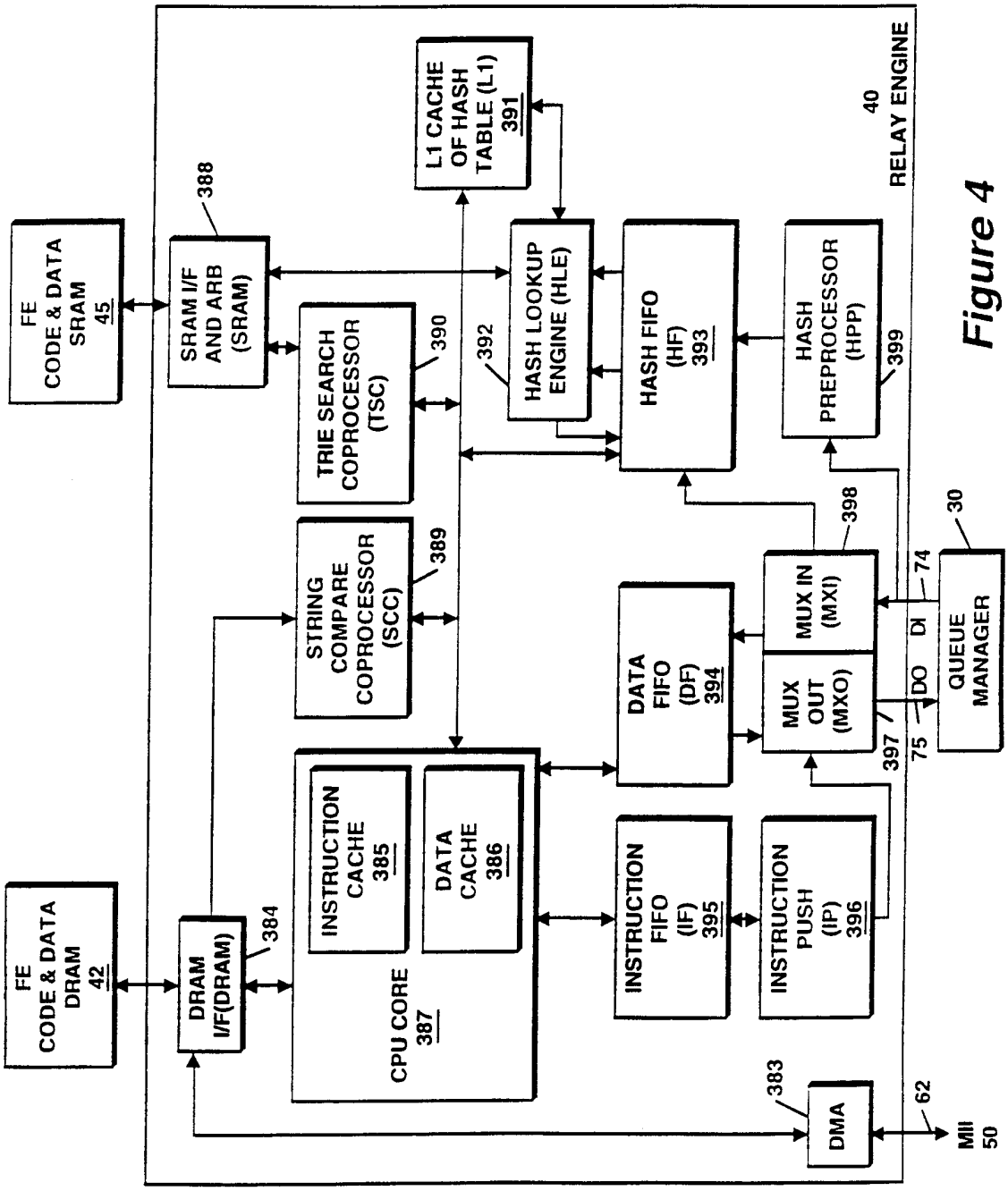

FE CODE & DATA DRAM 42
FE CODE & DATA SRAM 45
DRAM I/F(DRAM) 384
SRAM I/F AND ARB (SRAM) 388
L1 CACHE OF HASH TABLE (L1) 391
40 RELAY ENGINE
HASH LOOKUP ENGINE (HLE) 392
HASH FIFO (HF) 393
HASH PREPROCESSOR (HPP) 399
TRIE SEARCH COPROCESSOR (TSC) 390
STRING COMPARE COPROCESSOR (SCC) 389
INSTRUCTION CACHE 385
DATA CACHE 386
CPU CORE 387
DATA FIFO (DF) 394
MUX IN (MXI) 398
MUX OUT (MXO) 397
QUEUE MANAGER 30
DI 74
DO 75
INSTRUCTION FIFO (IF) 395
INSTRUCTION PUSH (IP) 396
DMA 383
MII 50 62

Figure 4

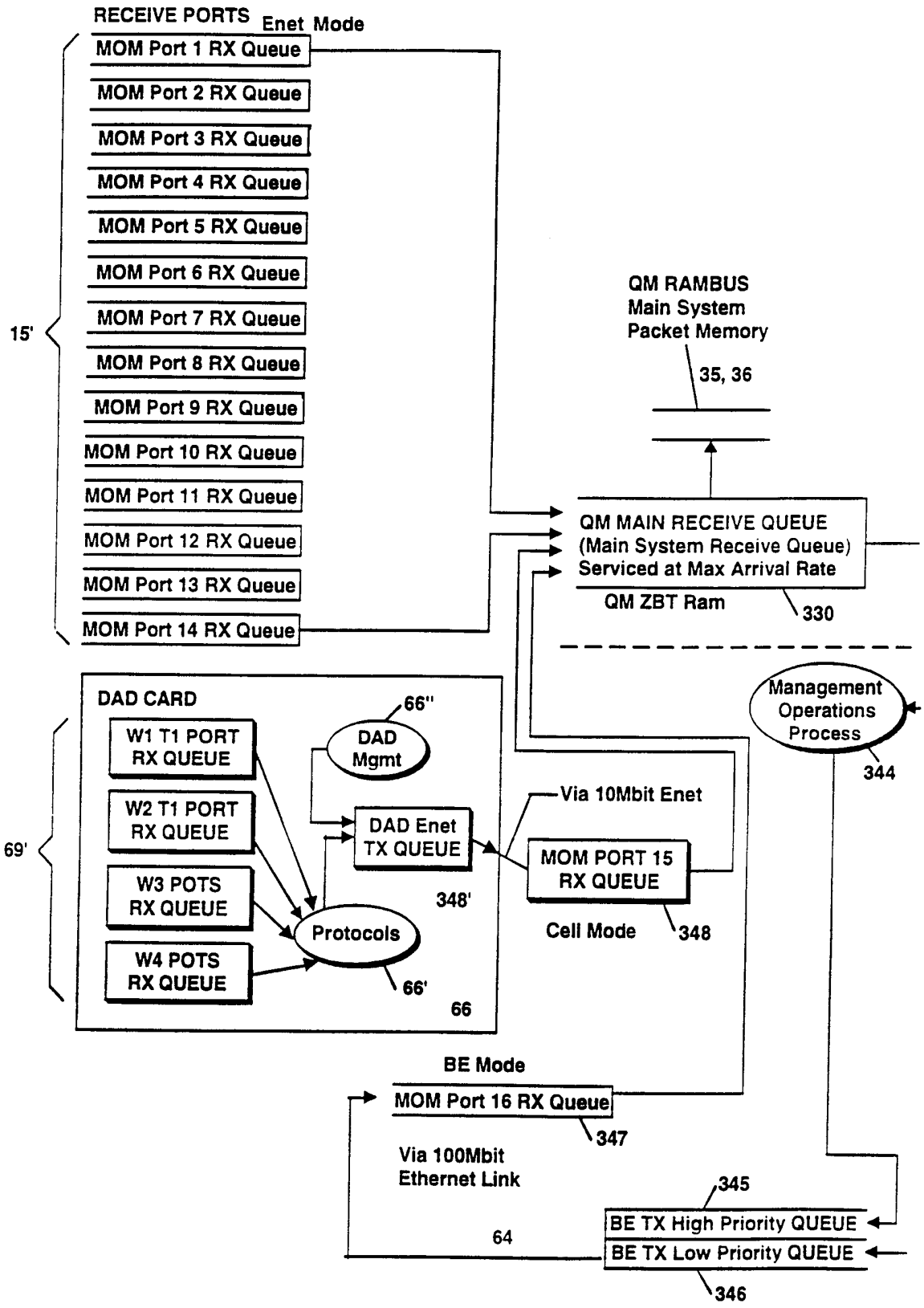

RECEIVE PORTS
Enet Mode
MOM Port 1 RX Queue
MOM Port 2 RX Queue
MOM Port 3 RX Queue
MOM Port 4 RX Queue
MOM Port 5 RX Queue
MOM Port 6 RX Queue
MOM Port 7 RX Queue
MOM Port 8 RX Queue
MOM Port 9 RX Queue
MOM Port 10 RX Queue
MOM Port 11 RX Queue
MOM Port 12 RX Queue
MOM Port 13 RX Queue
MOM Port 14 RX Queue
15'
QM RAMBUS
Main System
Packet Memory
35, 36
QM MAIN RECEIVE QUEUE
(Main System Receive Queue)
Serviced at Max Arrival Rate
QM ZBT Ram
330
DAD CARD
66"
DAD
Mgmt
W1 T1 PORT
RX QUEUE
W2 T1 PORT
RX QUEUE
W3 POTS
RX QUEUE
W4 POTS
RX QUEUE
69'
DAD Enet
TX QUEUE
348'
Protocols
66'
66
Via 10Mbit Enet
MOM PORT 15
RX QUEUE
Cell Mode
348
Management
Operations
Process
344
BE Mode
MOM Port 16 RX Queue
347
Via 100Mbit
Ethernet Link
345
BE TX High Priority QUEUE
64
BE TX Low Priority QUEUE
346

Figure 5

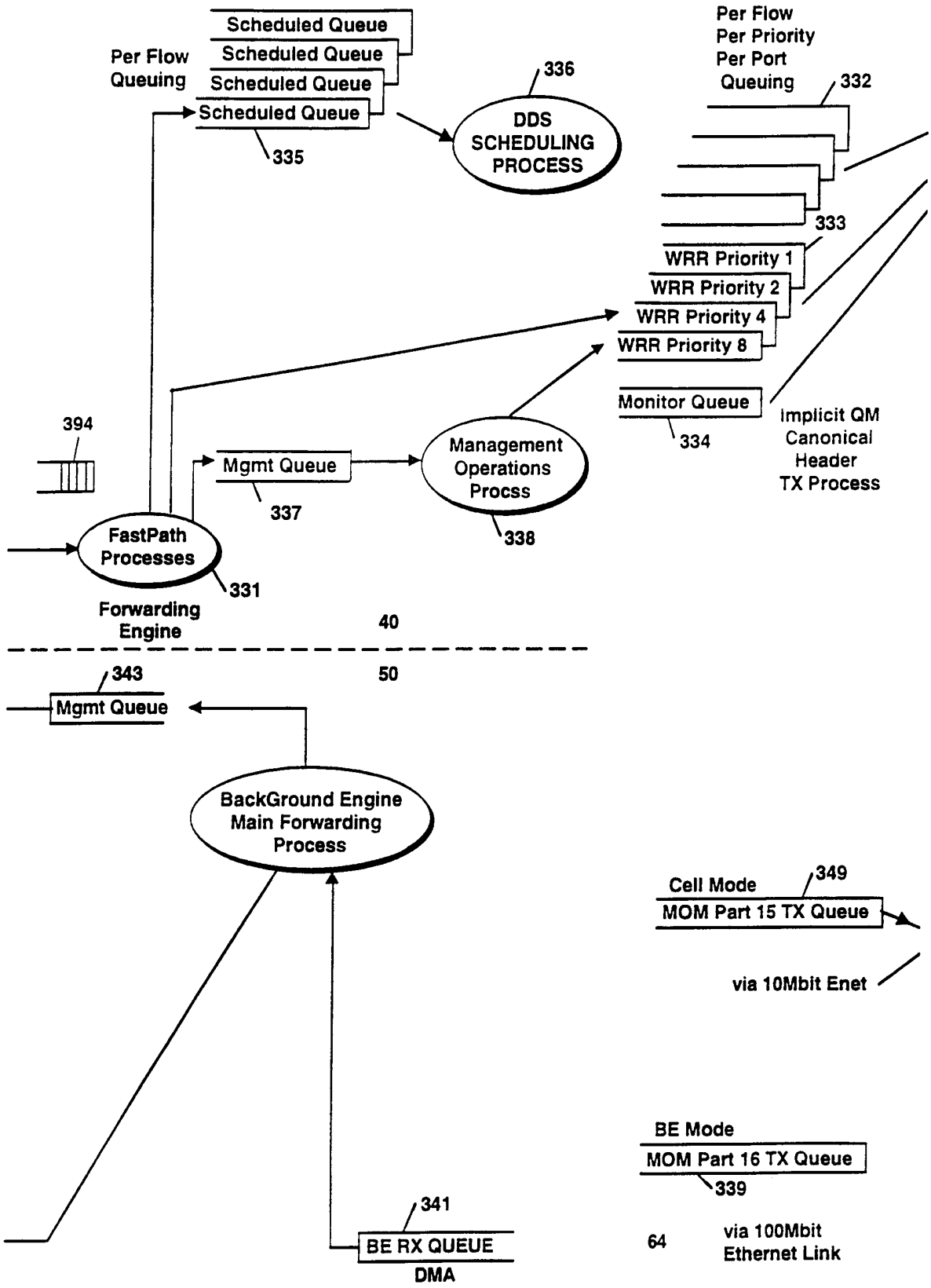

Scheduled Queue
Scheduled Queue
Scheduled Queue
Scheduled Queue
Per Flow Queuing
335
336
DDS SCHEDULING PROCESS
Per Flow
Per Priority
Per Port
Queuing
332
333
WRR Priority 1
WRR Priority 2
WRR Priority 4
WRR Priority 8
Monitor Queue
334
Implicit QM
Canonical
Header
TX Process
394
Mgmt Queue
337
Management Operations Procss
338
FastPath Processes
331
Forwarding Engine
40
50
343
Mgmt Queue
BackGround Engine Main Forwarding Process
Cell Mode
349
MOM Part 15 TX Queue
via 10Mbit Enet
BE Mode
MOM Part 16 TX Queue
339
64
via 100Mbit Ethernet Link
341
BE RX QUEUE
DMA

Figure 5A

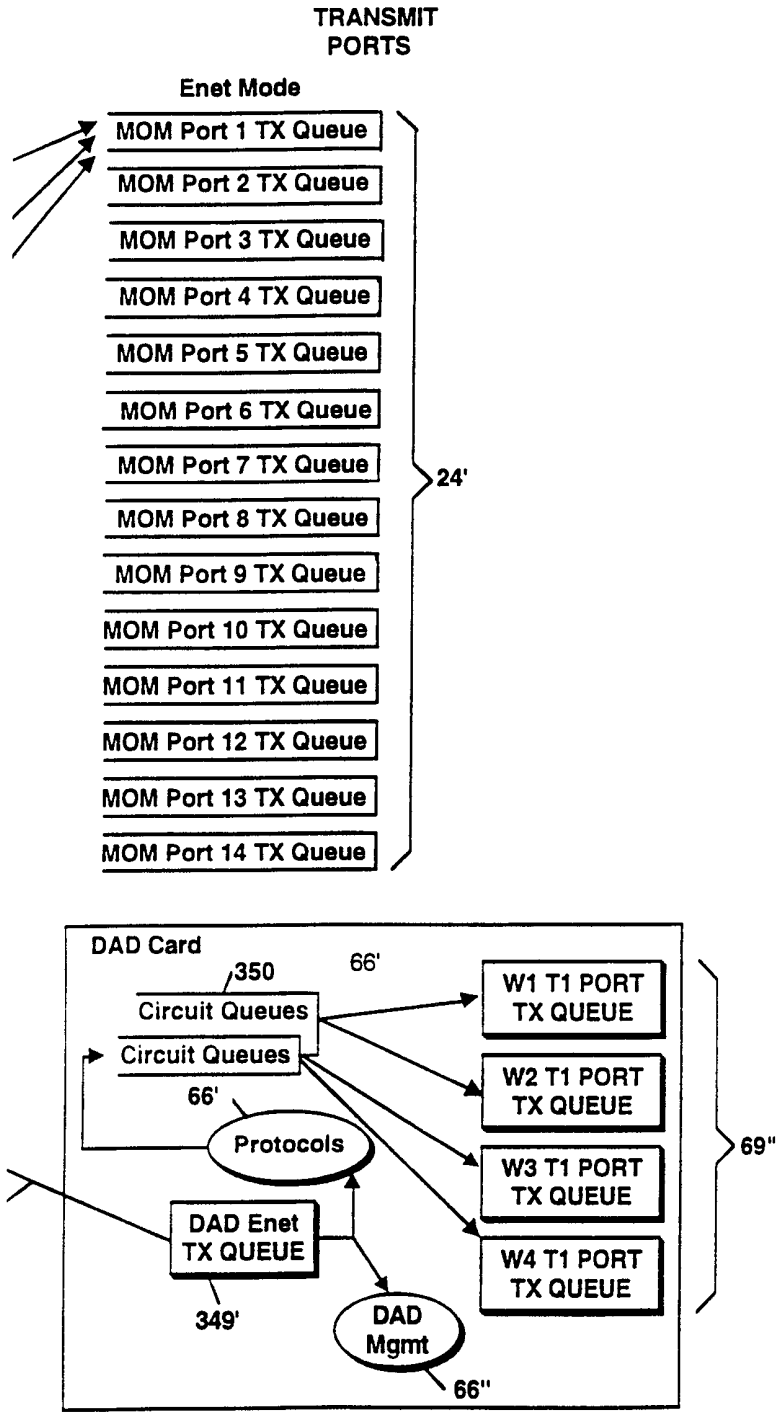

TRANSMIT PORTS
Enet Mode
MOM Port 1 TX Queue
MOM Port 2 TX Queue
MOM Port 3 TX Queue
MOM Port 4 TX Queue
MOM Port 5 TX Queue
MOM Port 6 TX Queue
MOM Port 7 TX Queue
MOM Port 8 TX Queue
MOM Port 9 TX Queue
MOM Port 10 TX Queue
MOM Port 11 TX Queue
MOM Port 12 TX Queue
MOM Port 13 TX Queue
MOM Port 14 TX Queue
24'
DAD Card
350
66'
Circuit Queues
Circuit Queues
66'
Protocols
DAD Enet TX QUEUE
349'
DAD Mgmt
66"
W1 T1 PORT TX QUEUE
W2 T1 PORT TX QUEUE
W3 T1 PORT TX QUEUE
W4 T1 PORT TX QUEUE
69"

Figure 5B

Figure 6A

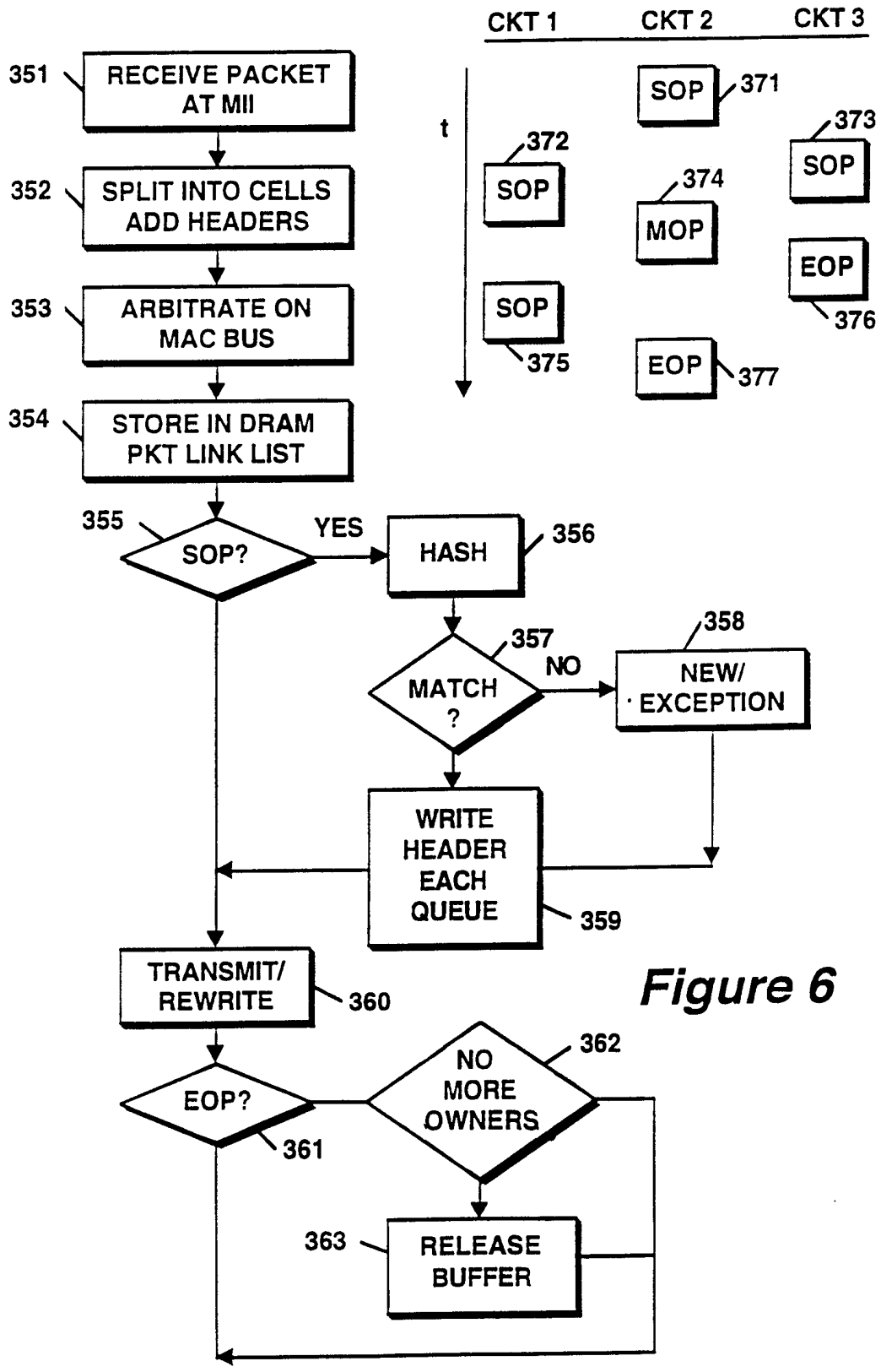
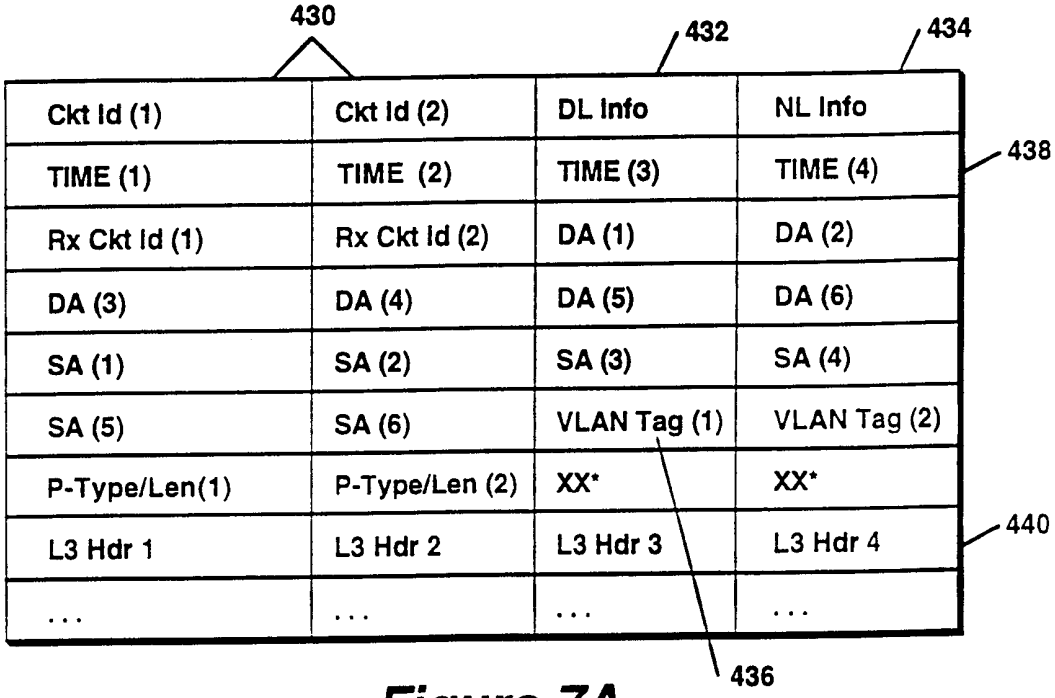


Figure 6



Ckt Id (1)	Ckt Id (2)	DL Info	NL Info
TIME (1)	TIME (2)	TIME (3)	TIME (4)
Rx Ckt Id (1)	Rx Ckt Id (2)	DA (1)	DA (2)
DA (3)	DA (4)	DA (5)	DA (6)
SA (1)	SA (2)	SA (3)	SA (4)
SA (5)	SA (6)	VLAN Tag (1)	VLAN Tag (2)
P-Type/Len(1)	P-Type/Len (2)	XX*	XX*
L3 Hdr 1	L3 Hdr 2	L3 Hdr 3	L3 Hdr 4
...	...	...	...

Figure 7A

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Vlan Tagged	CRC (1)	CRC (2)	XX	XX	PKT Format (1)	PKT Format (2)	PKT Format (3)

Figure 7B

Bit 6	Bit 5	DESCRIPTION
0	X	CRC Not Present - on Rcvd Pkts
1	X	CRC is Present - on Rcvd Pkts
0	0	Add CRC - on Xmit Pkts
0	1	Delete CRC - on Xmit Pkt
1	0	Modify CRC - on Xmit Pkts
1	1	Don't Change CRC - on Xmit Pkts

Figure 7C

Bit 2	Bit 2	Bit 2	DL HEADER FORMAT
0	0	0	Reserved
0	0	1	DIX Format
0	1	0	SAP Format
0	1	1	Novell Raw Format
1	0	0	SNAP Format
1	0	1	SNAP 1042 Format
1	1	0	Reserved
1	1	1	Reserved

Figure 7D

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OurMac(BridgeGroup)	OurMac(Port)	WellKnown DA	Protocol Kind (1)	Protocol Kind (2)	Protocol Kind (3)	Protocol Kind (4)	Protocol Kind (5)

Figure 8A

VALUE	PROTOCOL	DIX	RAW	SAP	SNAP 1042
0	Unknown				
1	IPv4	08-00			08-00
2	ARP	08-06			08-06
3	IPX	81-37	YES	E0	81-37
4	IPv6	86-DD			86-DD
5	AppleTalk	80-9B			
6	NetBeui	80-D5		F0	
7	DECnetIV	60-03			60-03
8	DECnet V			FE	
31	BLAZENET IPC				

Figure 8B

Byte 0	Byte 1	DL Header Format
XX	XX	DIX Format
SSAP	Ctl	SAP Format - Configured SAP
XX	XX	SAP Format - Unknown SAP
XX	XX	Novell Raw Format
SSAP	Ctl	SNAP Format
P-Type(1)	P-Type(2)	SNAP 1042 Format

Figure 8C

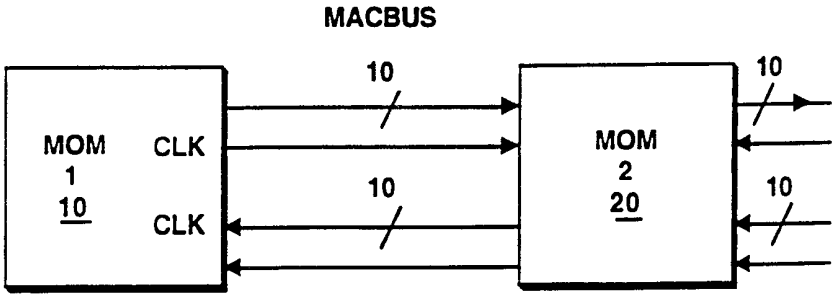


Figure 9

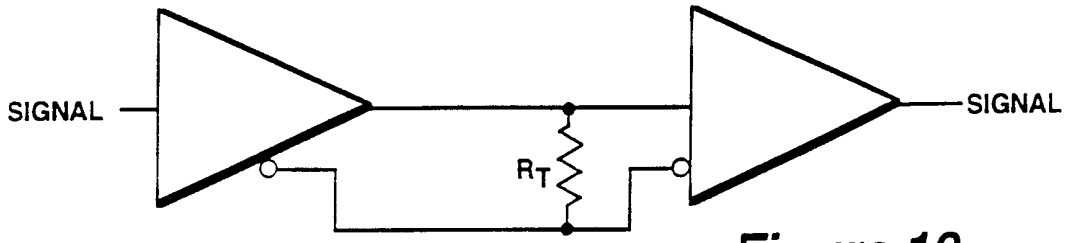


Figure 10

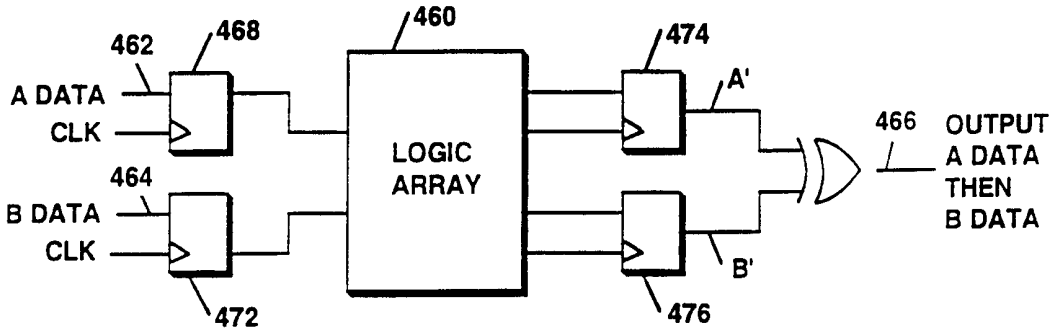


Figure 11

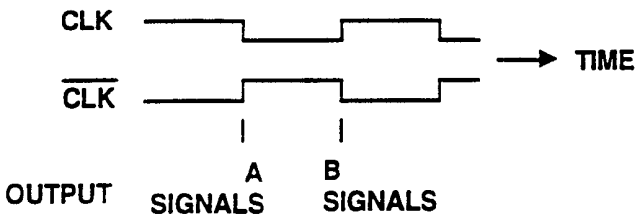


Figure 12

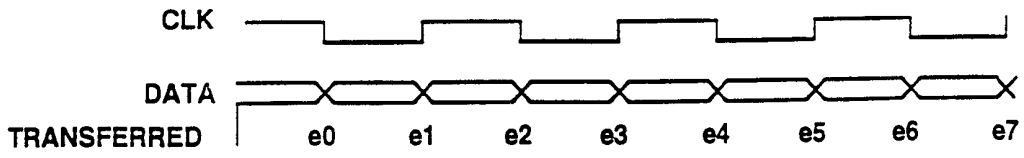


Figure 12A

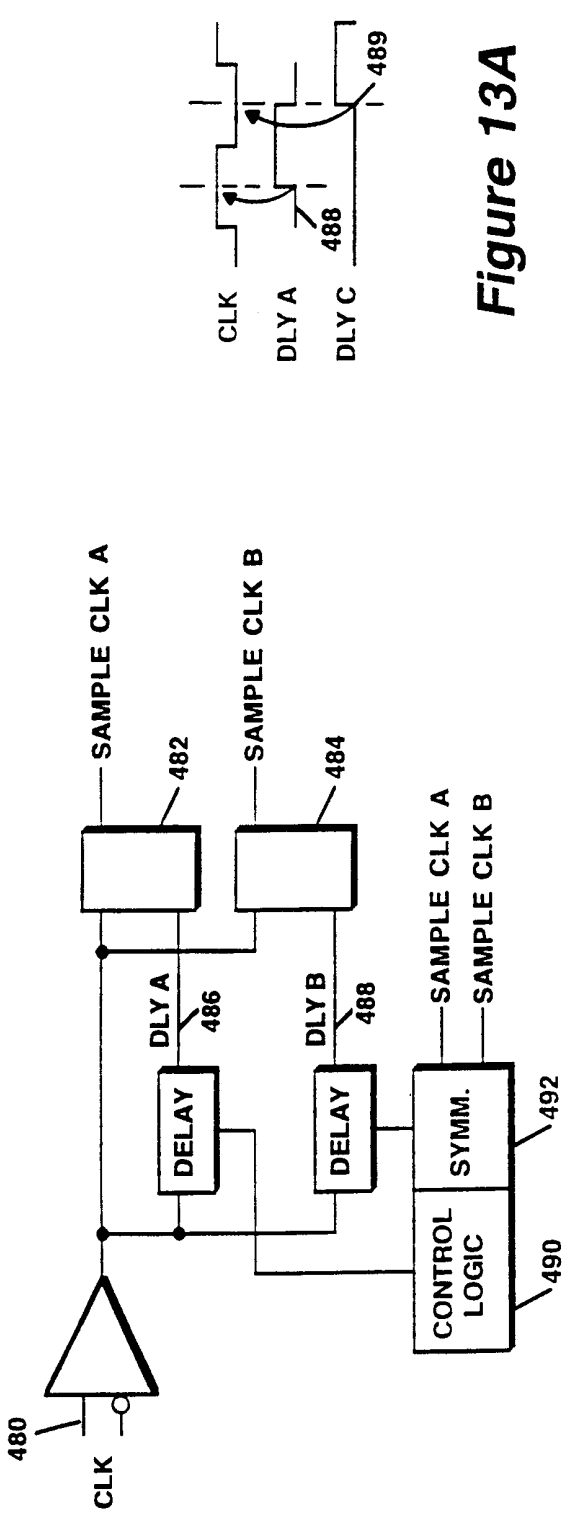


Figure 13

Figure 13A

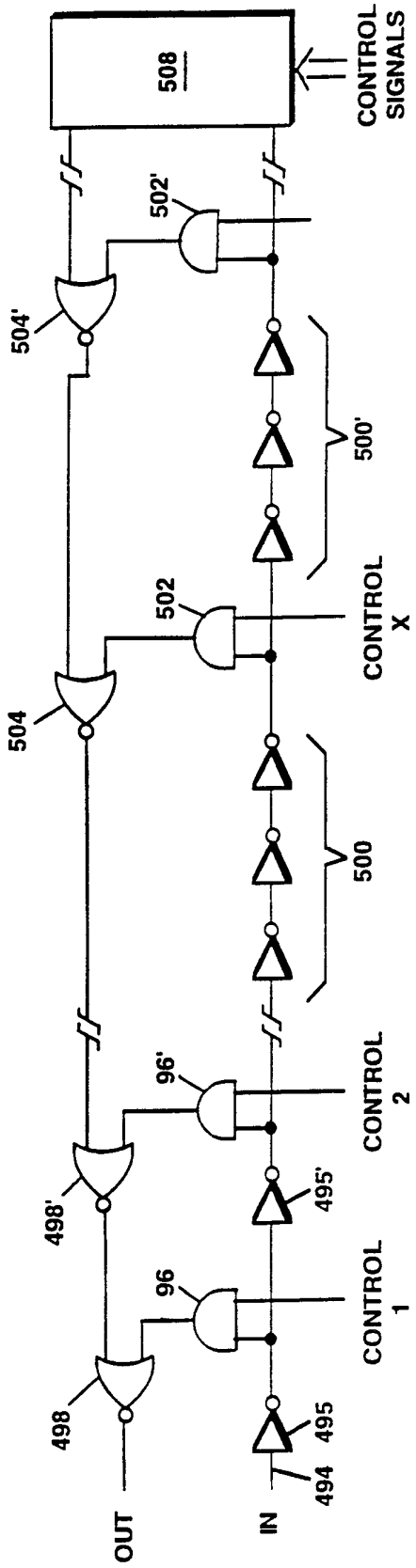


Figure 14

TABLE 1		e0	e2	e4	e6	FUNCTION
		0	0	0	0	IDLE
CONTROL BIT VALUE	610	1	0	0	0	FRAME 1
		1	1	0	0	FRAME 2
		1	1	1	0	FRAME 3

Figure 15

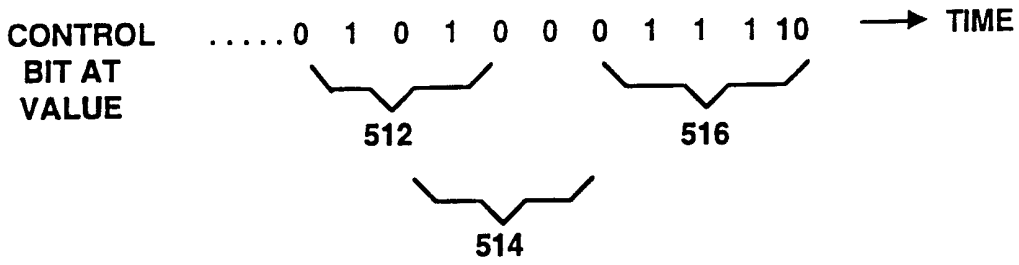


Figure 16

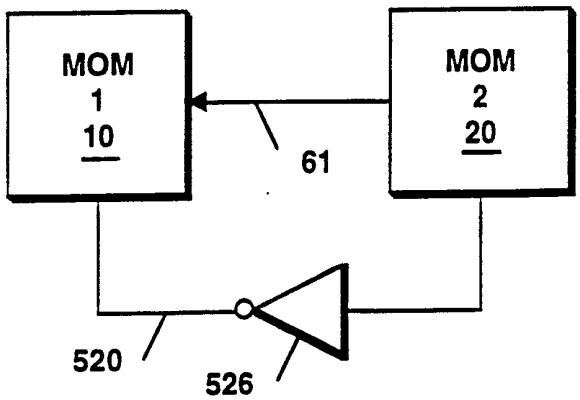


Figure 17

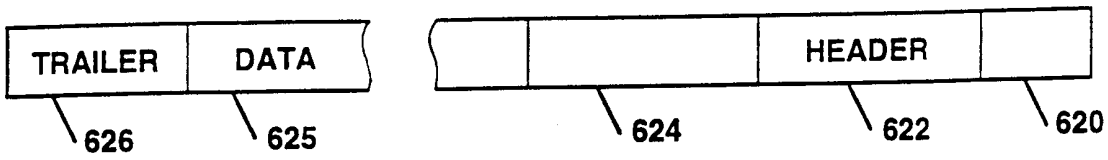


Figure 18

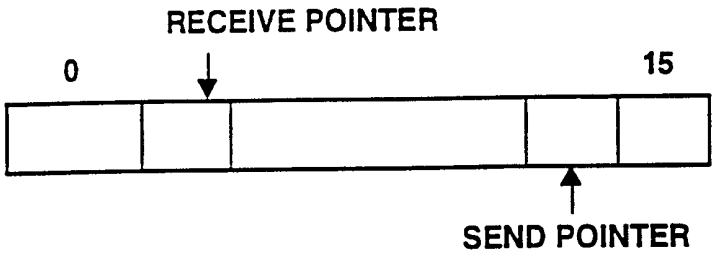


Figure 19

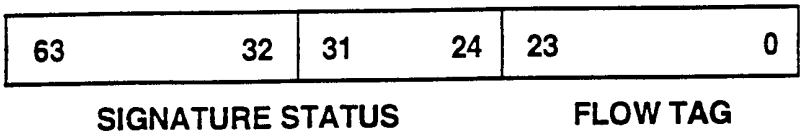


Figure 20A

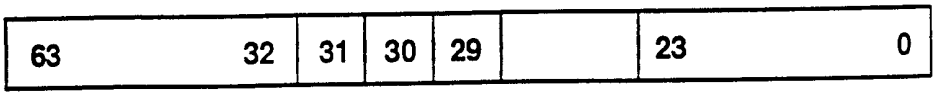


Figure 20B

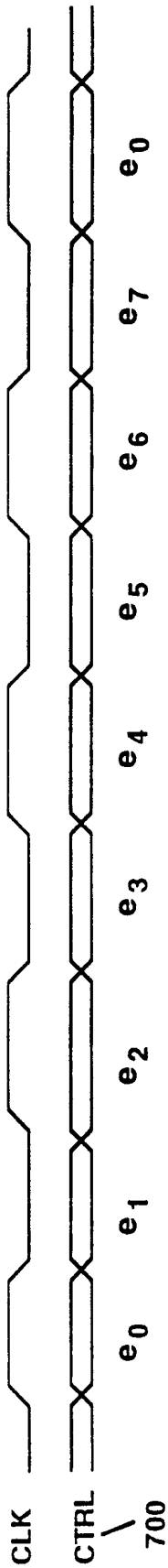


Figure 21

EVEN CTRL	MEANING	ODD CTRL	
e_0	= 1 ALWAYS	e_1	= 1 T_X CREDIT
e_2	= 1 START OF CELL	e_3	END CODE 0
e_4	= 1 START PKT	e_5	END CODE 1
e_6	= 0 ALWAYS	e_7	= 1 SHORT WORD

Figure 22



Figure 23

END	CODE	MEANING
1	0	
0	0	NOT END
0	1	END CELL
1	0	END PKT
1	1	END BAD PKT

Figure 25

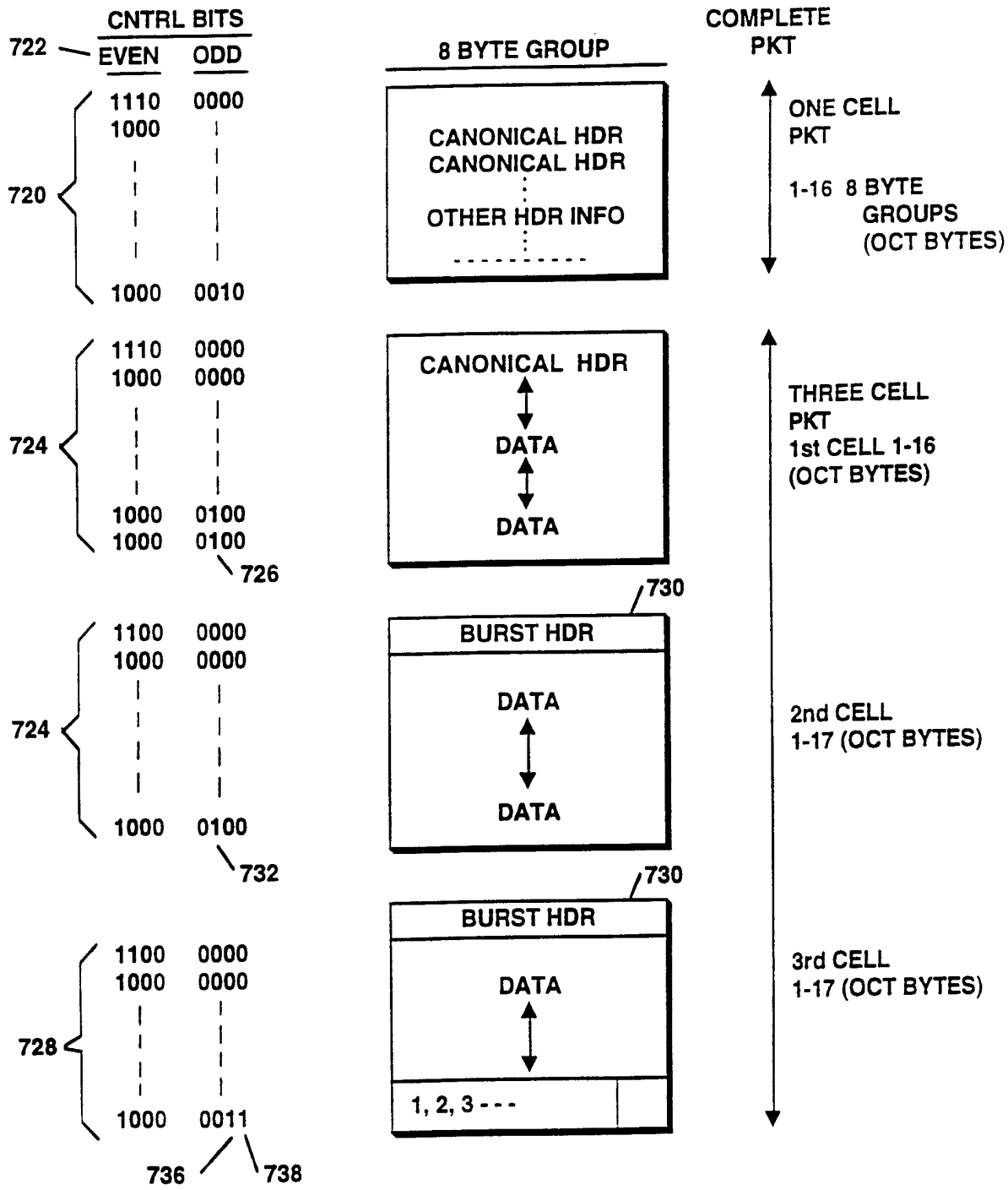

CNTRL BITS
722
EVEN
ODD
1110 0000
1000
720
1000 0010
1110 0000
1000 0000
724
1000 0100
1000 0100
726
1100 0000
1000 0000
724
1000 0100
732
1100 0000
1000 0000
728
1000 0011
736
738
8 BYTE GROUP
CANONICAL HDR
CANONICAL HDR
OTHER HDR INFO
CANONICAL HDR
DATA
DATA
730
BURST HDR
DATA
DATA
730
BURST HDR
DATA
1, 2, 3 - - -
COMPLETE PKT
ONE CELL PKT
1-16 8 BYTE GROUPS (OCT BYTES)
THREE CELL PKT
1st CELL 1-16 (OCT BYTES)
2nd CELL 1-17 (OCT BYTES)
3rd CELL 1-17 (OCT BYTES)

Figure 24

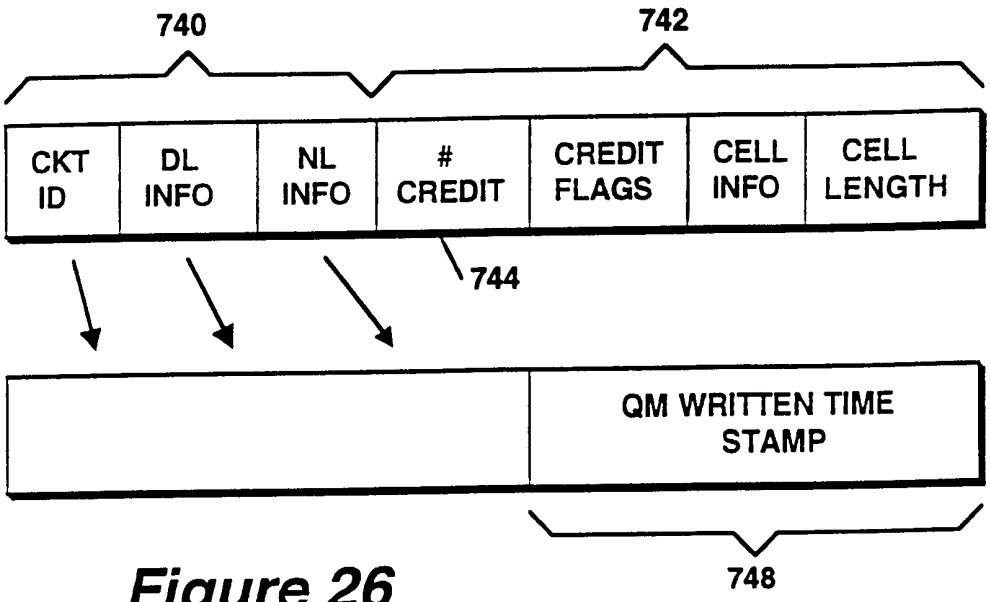


Figure 26

CELL INFO

7 - CELL ERROR
6 - PKT TIMEOUT
5 - FROM BAD PKG
4 - FROM MONITOR Q
3 - END CELL
2 - START CELL
1 - DATA CELL
0 - CREDIT CELL

Figure 27

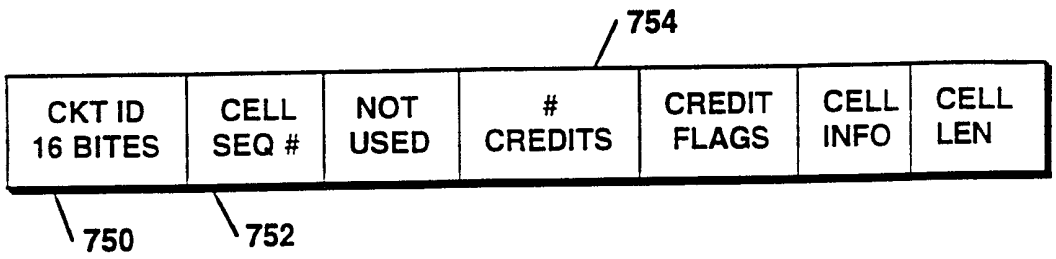


Figure 28



Figure 29B

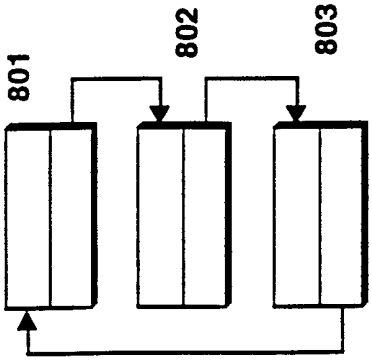


Figure 29

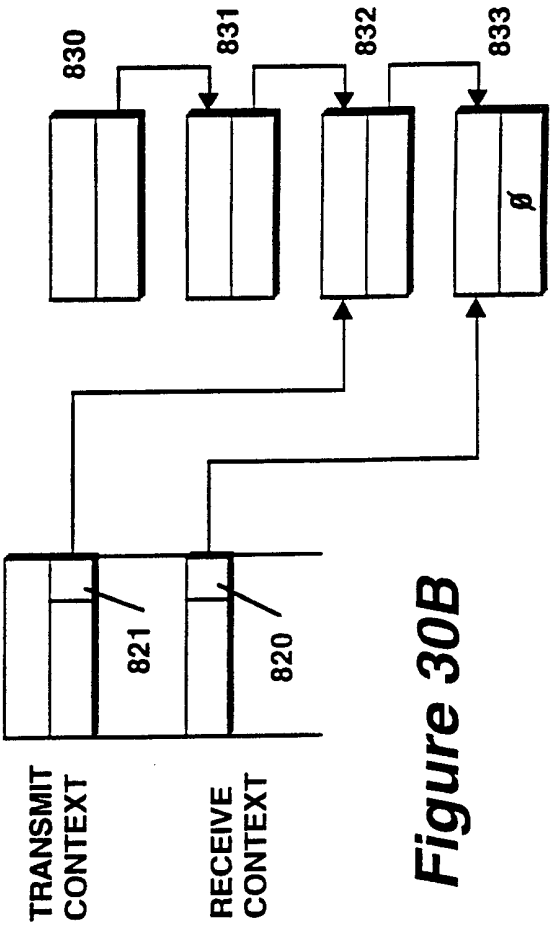


Figure 30B

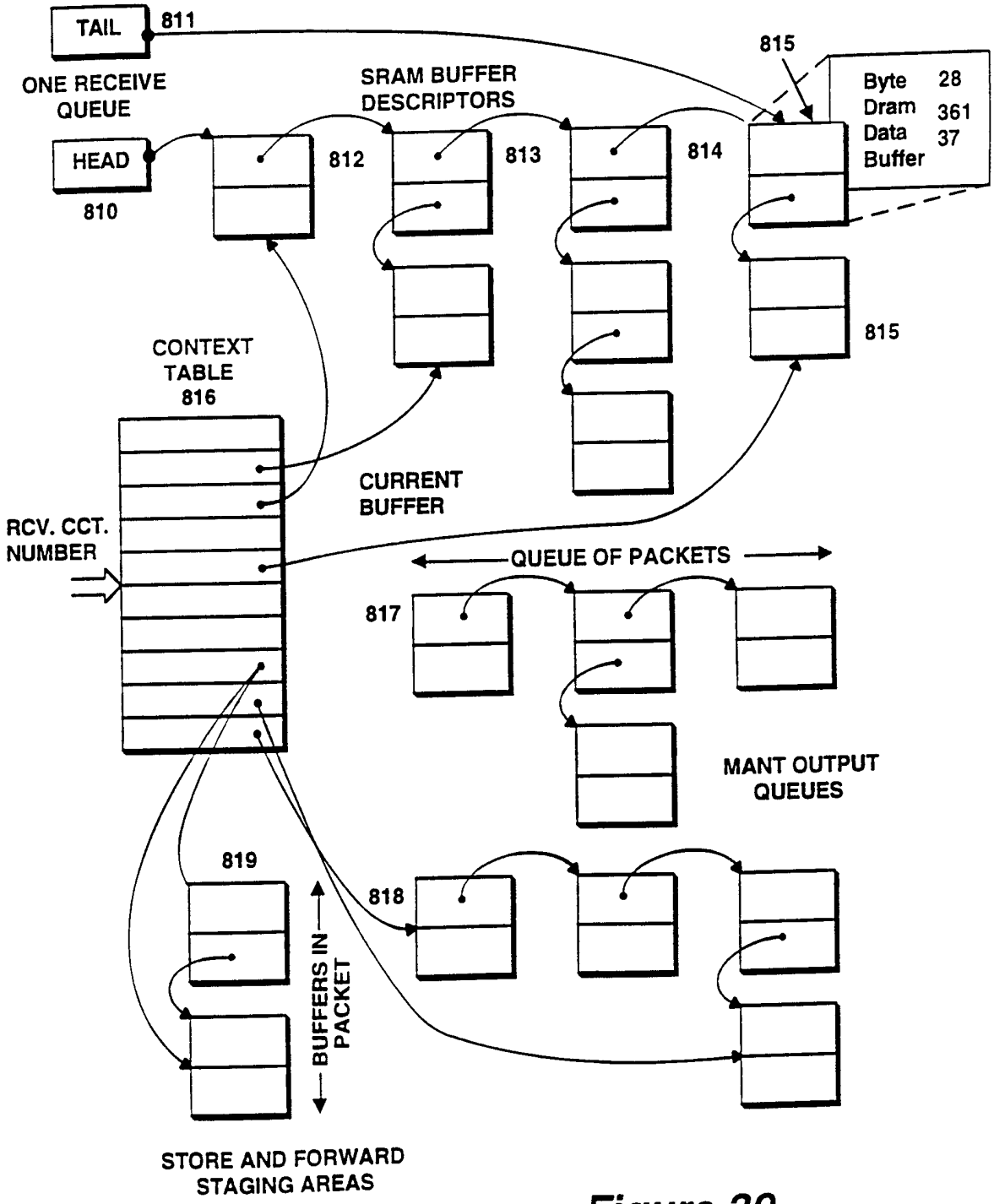

TAIL
811
ONE RECEIVE QUEUE
HEAD
810
SRAM BUFFER DESCRIPTORS
812
813
814
815
Byte 28
Dram 361
Data 37
Buffer
815
CONTEXT TABLE 816
CURRENT BUFFER
RCV. CCT. NUMBER
QUEUE OF PACKETS
817
MANT OUTPUT QUEUES
819
818
BUFFERS IN PACKET
STORE AND FORWARD STAGING AREAS

Figure 30

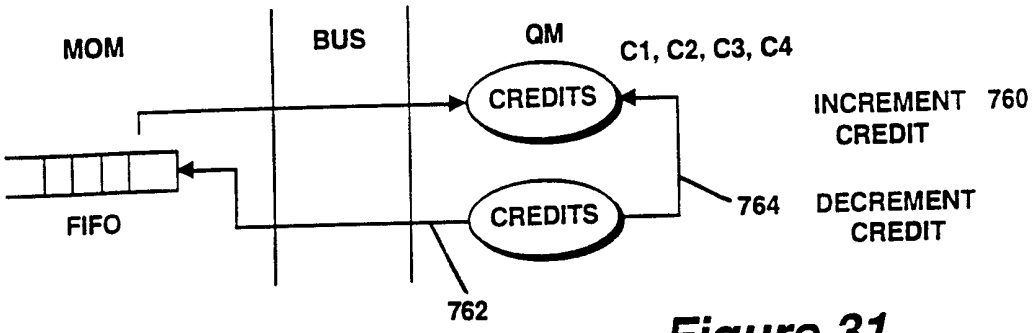


Figure 31

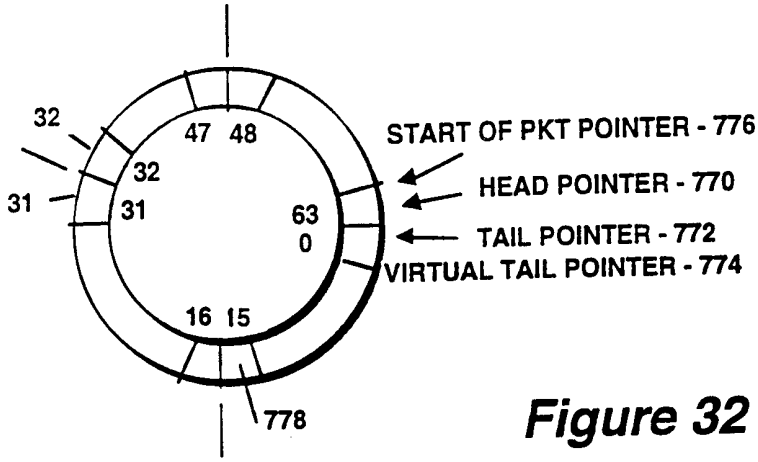


Figure 32

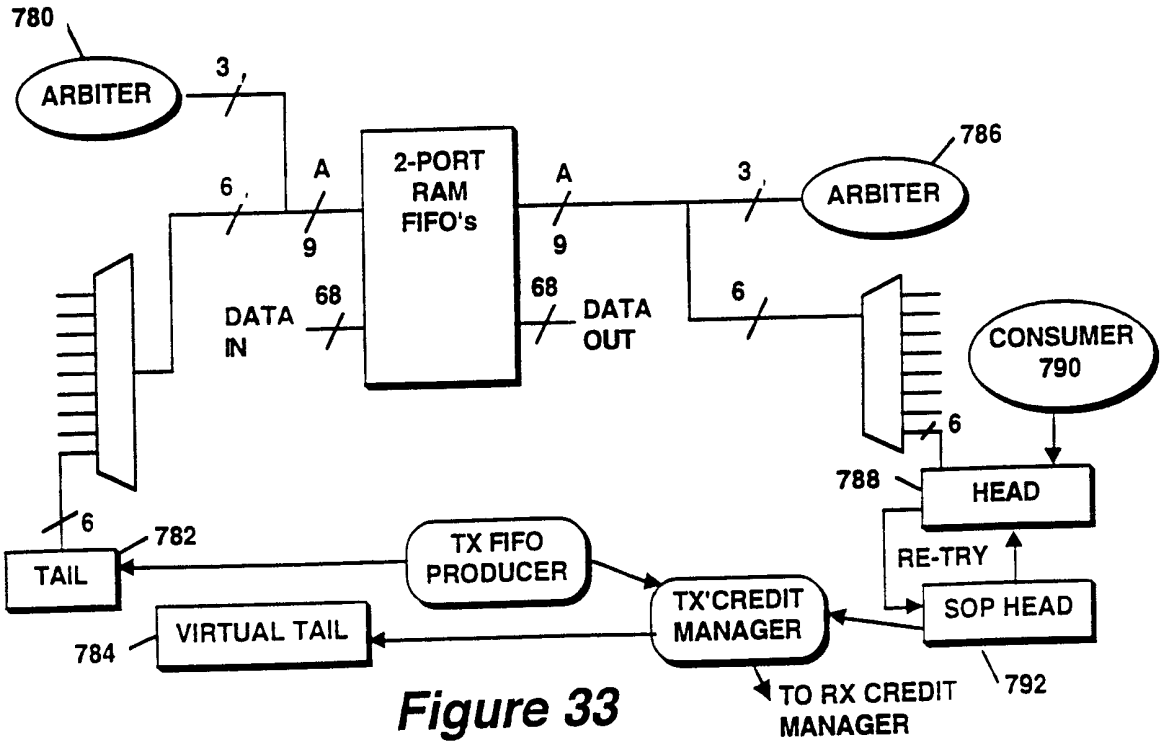


Figure 33

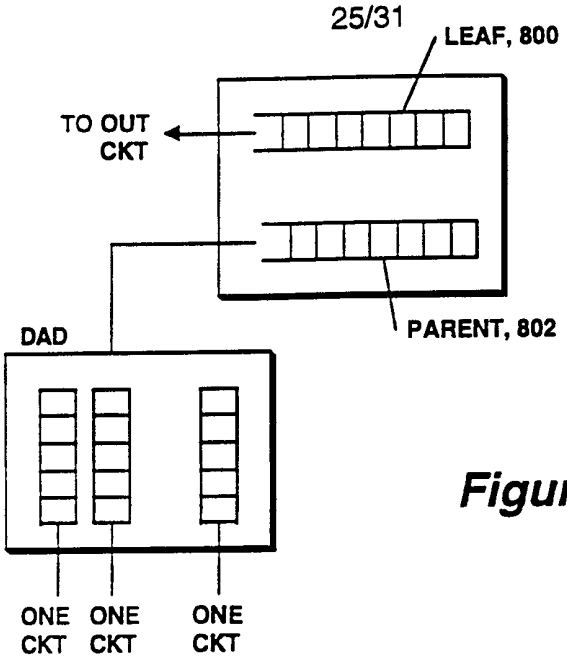


Figure 34

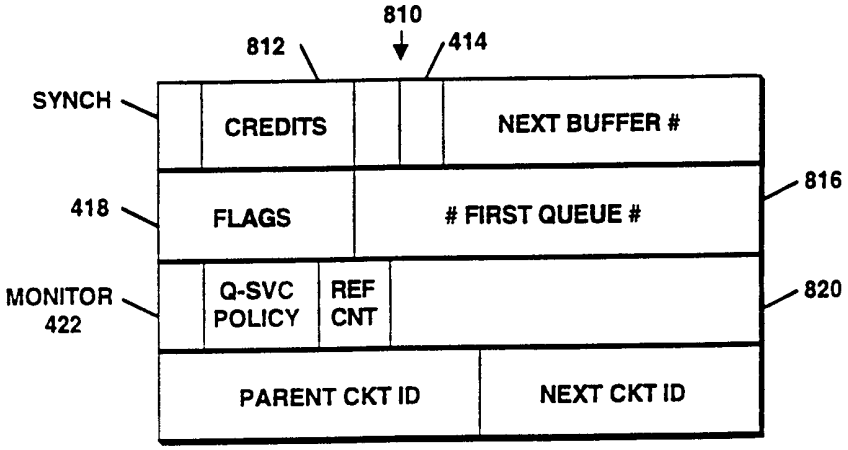


Figure 35

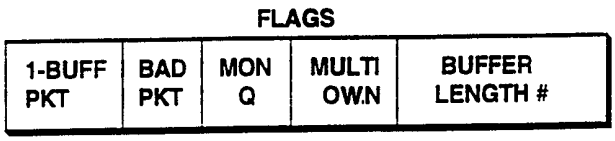
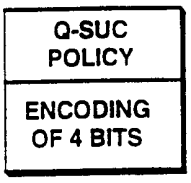


Figure 35A



1 QUEUE
2 STATIC Q's
3 WEIGHTED RRQ's (RR)
4 STATIC Q's
4 WEIGHTED RRQ's
2 STATIC/2RRQ's
8 STATIC
8 RR
4 STATIC/4RR
16 STATIC
16 RR
8 STATIC8/RR

Figure 35B

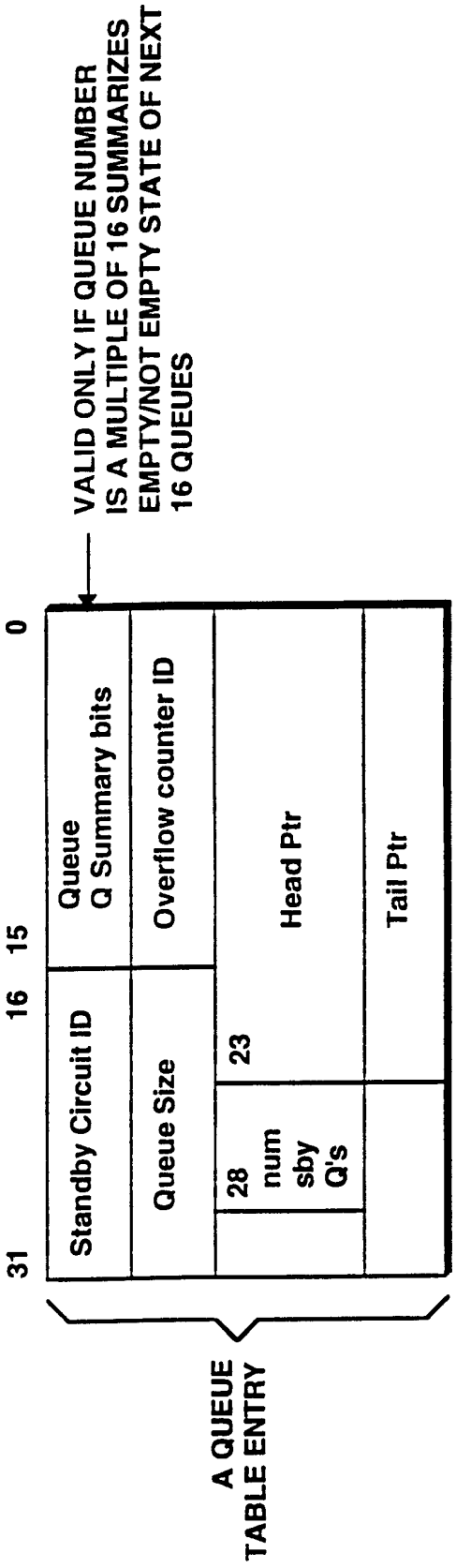

VALID ONLY IF QUEUE NUMBER
IS A MULTIPLE OF 16 SUMMARIZES
EMPTY/NOT EMPTY STATE OF NEXT
16 QUEUES
0
15
16
23
28
31
Queue
Q Summary bits
Overflow counter ID
Standby Circuit ID
Queue Size
Head Ptr
Tail Ptr
num
sby
Q's
A QUEUE
TABLE ENTRY

Figure 36

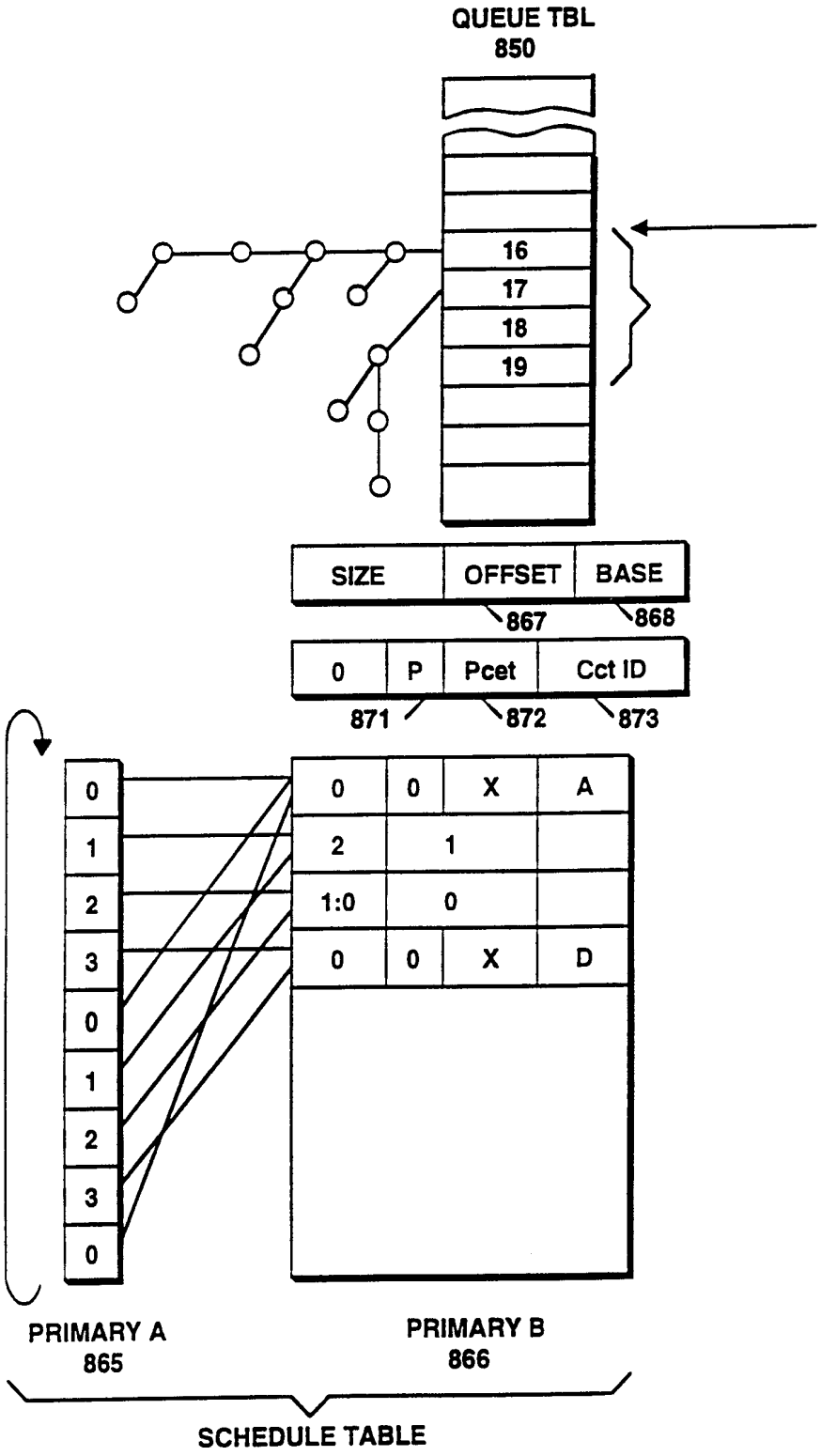

QUEUE TBL
850
16
17
18
19
SIZE
OFFSET
BASE
867
868
0
P
Pcet
Cct ID
871
872
873
0
1
2
3
0
1
2
3
0
0
0
X
A
2
1
1:0
0
0
0
X
D
PRIMARY A
865
PRIMARY B
866
SCHEDULE TABLE

Figure 37

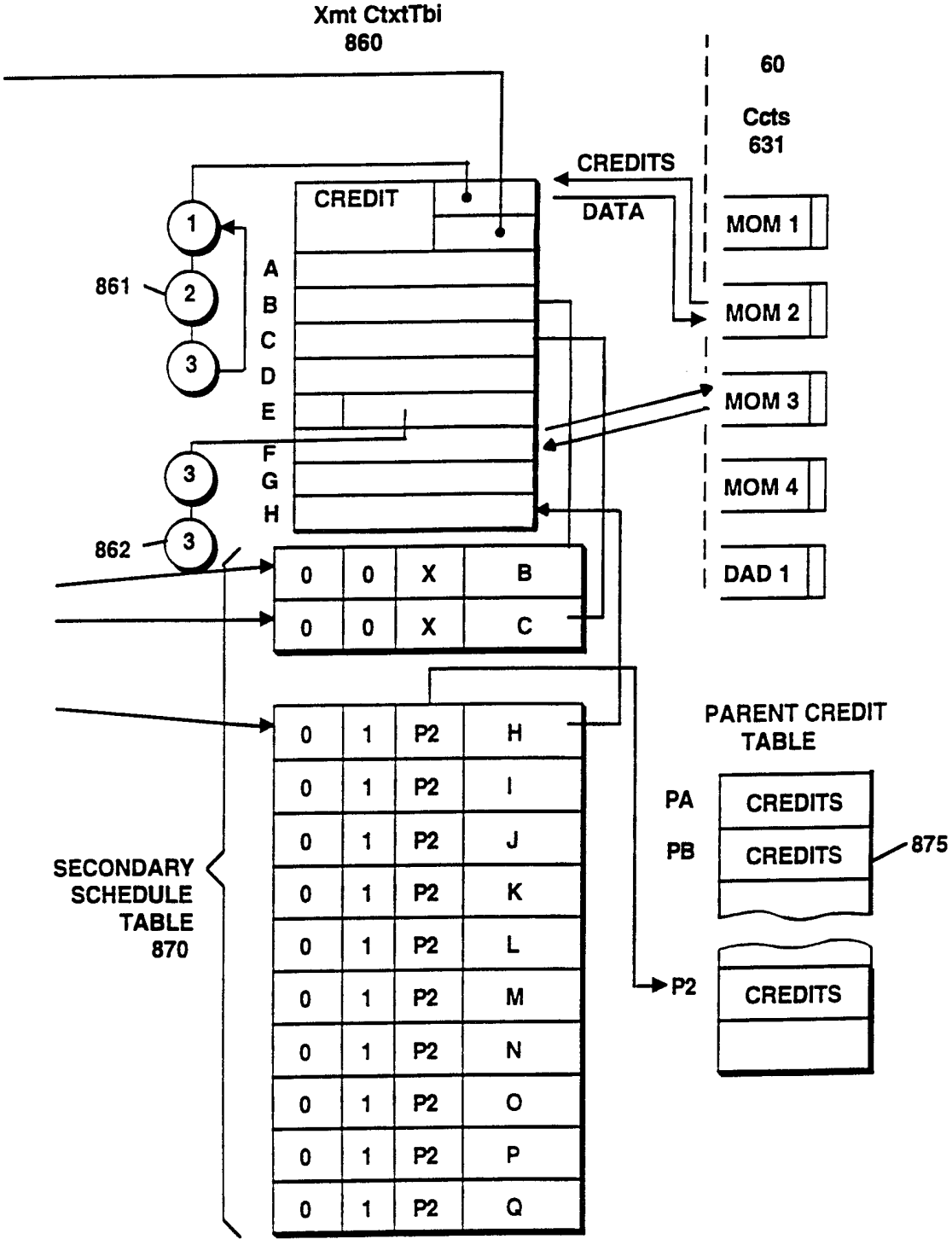

Xmt CtxtTbi
860
60
Ccts
631
CREDITS
DATA
CREDIT
MOM 1
MOM 2
MOM 3
MOM 4
DAD 1
861
862
A
B
C
D
E
F
G
H
0 0 X B
0 0 X C
0 1 P2 H
0 1 P2 I
0 1 P2 J
0 1 P2 K
0 1 P2 L
0 1 P2 M
0 1 P2 N
0 1 P2 O
0 1 P2 P
0 1 P2 Q
SECONDARY SCHEDULE TABLE 870
PARENT CREDIT TABLE
PA
PB
CREDITS
CREDITS
875
P2
CREDITS

Figure 37A

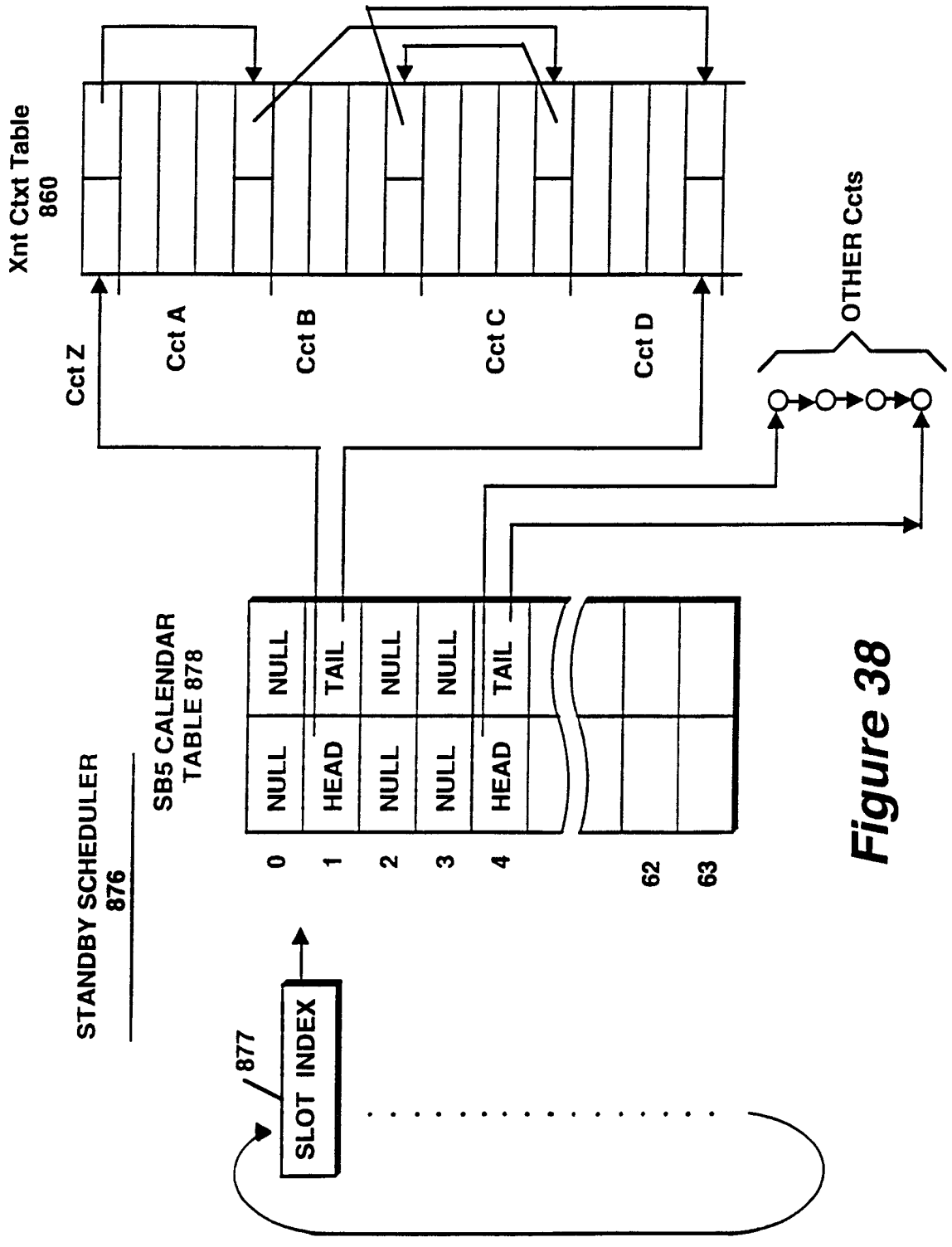

Xnt Ctxt Table
860
Cct Z
Cct A
Cct B
Cct C
Cct D
OTHER Ccts
STANDBY SCHEDULER
876
SB5 CALENDAR
TABLE 878
NULL
NULL
TAIL
NULL
NULL
TAIL
NULL
HEAD
NULL
NULL
HEAD
0
1
2
3
4
62
63
877
SLOT INDEX

Figure 38

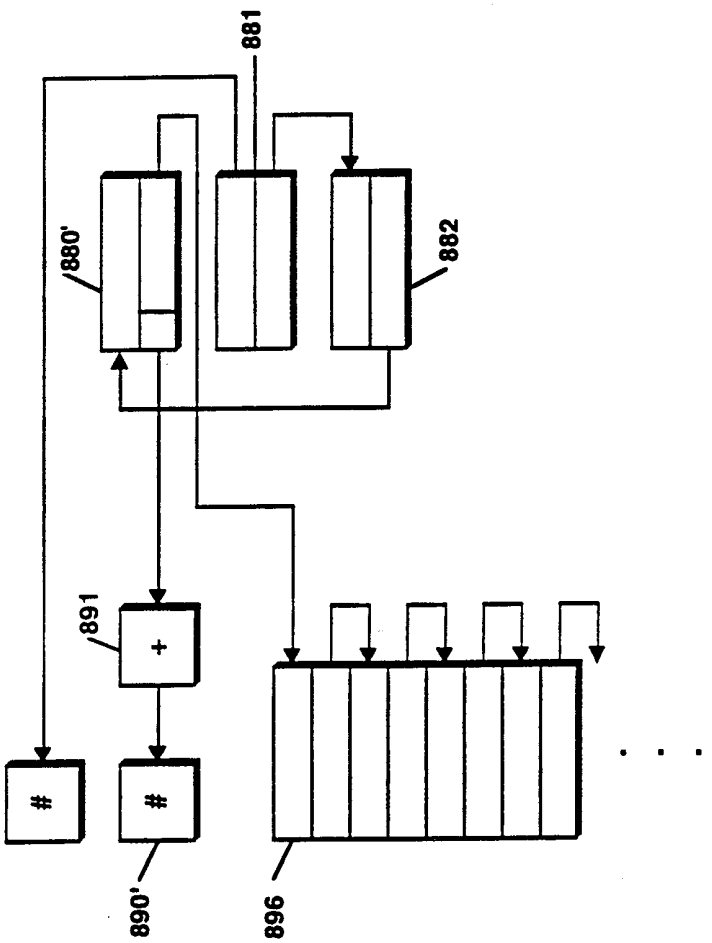


Figure 39B

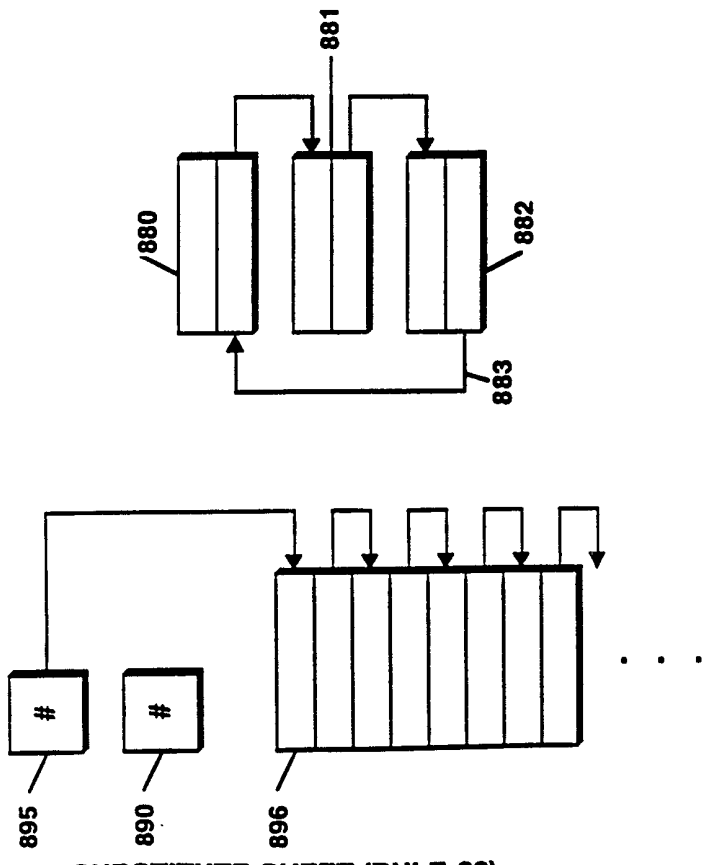


Figure 39A

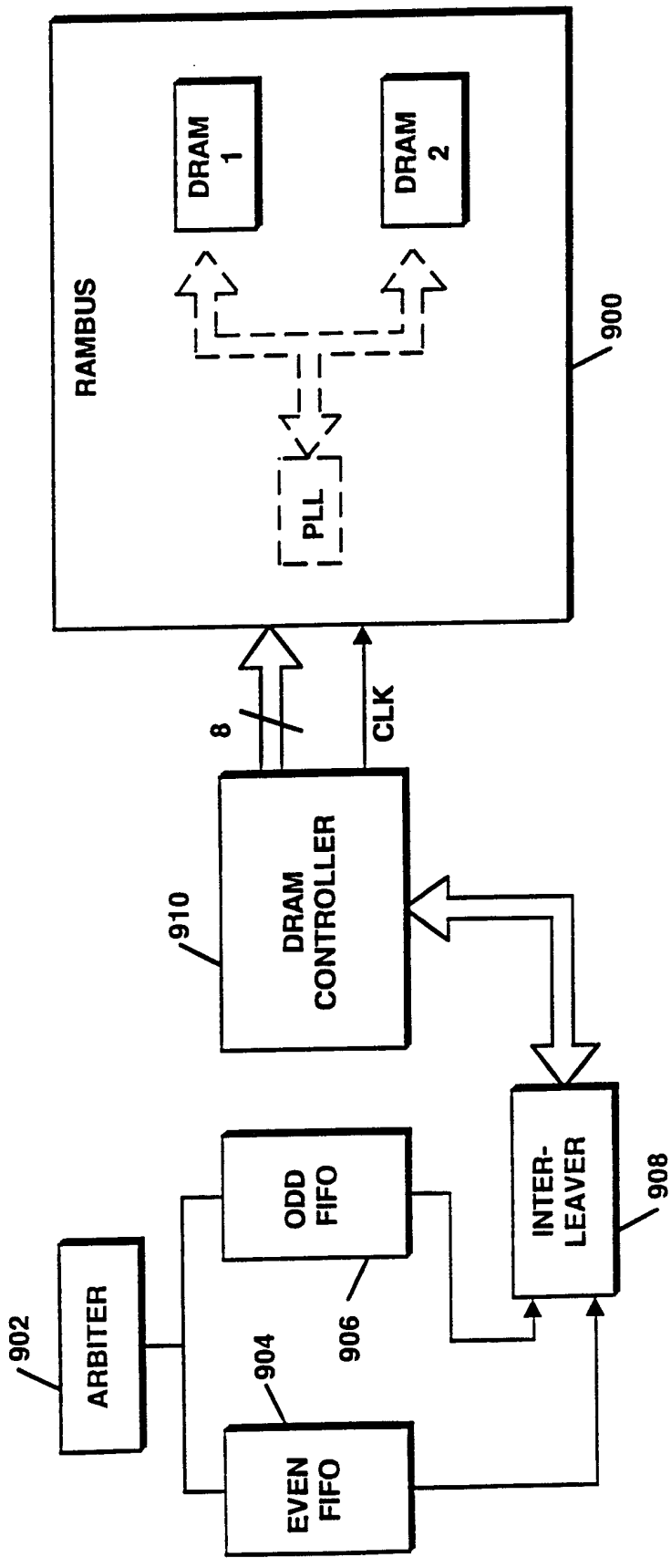

RAMBUS
DRAM 1
DRAM 2
PLL
900
8
CLK
DRAM CONTROLLER
910
INTER-LEAVER
908
ODD FIFO
906
EVEN FIFO
904
ARBITER
902

Figure 40